APPENDIX A : An EMBODIMENT OF RDL GRAMMAR

This section describes an embodiment of the RDL in a format that can be directly passed to a *yacc* tool. The bold letters are grammar keywords and the rest are the grammar rules. The grammar describes the allowed syntax for describing the target architecture.

RDL Grammar:

```
Top : Top Defs
    | Defs

Defs : UnitDef
     | ResourceDef
     | FunctionalityDef
     | ParameterDef

ParameterDef : PARAMETER Name INTEGER

Unitdef : UNITDEF Name
          {
            UnitBody
          }

UnitBody : UnitInstantiationsList StatementList

UnitInstantiationsList : UnitInstantiationsList UnitInstantiations
                       | UnitInstantiations

UnitInstantiations : UnitInstance
                   | ResourceInstance

ResourceInstance : RESOURCE Name ArrayNameList
```

```
ArrayNameList : ArrayNameList , ArrayName
              | ArrayName

ArrayName : ArrayName | ArithExpr |
          | ArrayName ( ArithExpr )
          | Name

UnitInstance : UNIT Name ArrayNameList

BasicStmt : RconnectStmt
          | UseStmt

RconnectStmt : RCONNECT( ArrayNameList ) ;

UseStmt : USES( ArrayNameList ) ;

FunctionalityDef : FUNCTIONALITYDEF Name
                 {
                     FuncDeclarationList
                     FuncConnectionList
                 }

FunctionDeclarationList : FunctionDeclarationList FuncDeclaration
                        | FuncDeclaration

FunctionConnectionList : FunctionConnectionList FuncConnection
                       | FuncConnection

NameList : NameList , Name
         | Name
FuncDeclaration : INPUT NameList ;
                | INPUT_OPT NameList ;
                | OUTPUT NameList ;
                | OUTPUT_OPT NameList ;
                | Name NameList ;

FuncConnection : DCONNECT( Name , Name ) ;
```

```
ResourceDef : RESOURCEDEF Name
                {
                Functionality_Supported
                Wrapper
                }

Functionality_Supported : FUNCTIONALITY NameList ;

Wrapper : Wrapper Attribute
          | Attribute

Attribute : ATTRIBUTE Name
            {
                AttributeBody
            }

AttributeBody : Cminus
                | FuncBasedAttributeList

FuncBasedAttributeList : FuncBasedAttributeList FuncBasedAttribute
                         | FuncBasedAttribute

FuncBasedAttribute : < Name > Cminus

Cminus : { CminusDeclrList StatementList }
         | { CminusDeclrList }
         | { StatementList }

CminusDeclrList : CminusDeclrList CminusDeclr
                  | CminusDeclr

CminusDeclr : float NameList
              | int NameList
              | table NameList

Statement : IfStmt
```

```
                | ForStmt
                | CminusBasicStmt
                | BasicStmt

StatementList : StatementList Statement
                | Statement

CminusBasicStmt : AssignExpr
                  | ArithExpr

IfStmt : if(  LogicalExpr )
         {
            StatementList
         }
         else
         {
            StatementList
         }
          | if(  LogicalExpr )
         {
            StatementList
         }

ForStmt : for( AssignExpr ; LogicalExpr ; AssignExpr )
          {
             StatementList
          }

AssignExpr : Name =  ArithExpr

ArithExpr : ArithExpr +  ArithExpr
            | ArithExpr -  ArithExpr
            | ArithExpr *  ArithExpr
            | ArithExpr /  ArithExpr
            | ArithExpr %  ArithExpr
            | (ArithExpr)
            | DataType
```

LogicalExpr : LogicalExpr == LogicalExpr
| LogicalExpr != LogicalExpr
| LogicalExpr < LogicalExpr
| LogicalExpr <= LogicalExpr
| LogicalExpr >= LogicalExpr
| LogicalExpr > LogicalExpr
| LogicalExpr || LogicalExpr
| LogicalExpr && LogicalExpr
| LogicalExpr
| ArithExpr

DataType : INTEGER
| FLOATINGPOINT
| STRING
| CompositeName

CompositeName : CompositeName -> ArrayName
| ArrayName

Name : IDENTIFER

APPENDIX B

An RDL specification for Xilinx Virtex2 XC2V250

```
//*********************************************************************
***
//*     COPYRIGHT (C), 2001
//*
//*   Company ..................... AccelChip Inc.
//*       999 Plaza Drive, Suite 340, Schaumburg, IL 60173
//*
//*   Program ..................... Description of the Xilinx
//*                                 Virtex2 XC2V250
//*                                 architecture in ACCEL RDL
//*
//*
//*********************************************************************
***/

////////////////////////////////////////////////////////////////////////
/
// This is the RDL file for Xilinx's Virtex2 device XC2V250
//
// This device has 24 SelectRAM memory blocks with storage for 432 Kb
//
// of data. The location properties use the following form :
//
// LOC = RAMB16_X#Y#. It has 24 multipliers arranged in 4 columns with
//
// 6 multipliers per column. The location properties use the
//
// following form : LOC = MULT18X18_X#Y#
//
// The Logic Cells are arranged in a 24 x 16 grid
//
////////////////////////////////////////////////////////////////////////
/

PARAMETER NUM_MULT_COLUMNS 4;
PARAMETER NUM_EMBED_MULT_PER_COLUMN 6;
PARAMETER NUM_RAM 24;
PARAMETER NUM_LOGIC_ROWS 24;
PARAMETER NUM_LOGIC_COLUMNS 16;

RESOURCEDEF GLOBAL
{
//////////////////////////////////////////
//

FUNCTIONALITY GLOBALDECLARATIONS;

ATTRIBUTE INSTANTIATION
   {
```

```
//////////////////////////////////////////
// Component Declaration for the Xilinx  //
// embedded multipliers                  //
//////////////////////////////////////////

attribute +     "void MULT18X18(input wire[18] A,";
attribute +         "input wire[18] B,";
attribute +         "output wire[36] P";
attribute +         "); \n";

//////////////////////////////////////////
// Component instantiation for the Xilinx//
// Block SelectRam with Dual Port        //
// A:16,384 x 1 bit and B:16,384 x 1 bit //
//////////////////////////////////////////

attribute +     "void RAMB16_S1_S1(input wire[1] DIA,";
attribute +         "input wire[14] ADDRA,";
attribute +         "input wire[1] ENA,";
attribute +         "input wire[1] WEA,";
attribute +         "input wire[1] SSRA,";
attribute +         "input wire[1] CLKA,";
attribute +         "output wire[1] DOA,";

attribute +         "input wire[1] DIB,";
attribute +         "input wire[14] ADDRB,";
attribute +         "input wire[1] ENB,";
attribute +         "input wire[1] WEB,";
attribute +         "input wire[1] SSRB,";
attribute +         "input wire[1] CLKB,";
attribute +         "output wire[1] DOB";

attribute +         "); \n";

//////////////////////////////////////////
// Component instantiation for the Xilinx//
// Block SelectRam with Dual Port        //
// A:8192 x 2 bits and B:8192 x 2 bits   //
//////////////////////////////////////////

attribute +     "void RAMB16_S2_S2(input wire[2] DIA,";
attribute +         "input wire[13] ADDRA,";
attribute +         "input wire[1] ENA,";
attribute +         "input wire[1] WEA,";
attribute +         "input wire[1] SSRA,";
attribute +         "input wire[1] CLKA,";
attribute +         "output wire[2] DOA,";

attribute +         "input wire[2] DIB,";
attribute +         "input wire[13] ADDRB,";
attribute +         "input wire[1] ENB,";
attribute +         "input wire[1] WEB,";
attribute +         "input wire[1] SSRB,";
```

```
attribute +         "input wire[1] CLKB,";
attribute +         "output wire[2] DOB";

attribute +         "); \n";

//////////////////////////////////////////
// Component instantiation for the Xilinx//
// Block SelectRam with Dual Port        //
// A:4096 x 4 bits and B:4096 x 4 bits   //
//////////////////////////////////////////

attribute +     "void RAMB16_S4_S4(input wire[4] DIA,";
attribute +         "input wire[12] ADDRA,";
attribute +         "input wire[1] ENA,";
attribute +         "input wire[1] WEA,";
attribute +         "input wire[1] SSRA,";
attribute +         "input wire[1] CLKA,";
attribute +         "output wire[4] DOA,";

attribute +         "input wire[4] DIB,";
attribute +         "input wire[12] ADDRB,";
attribute +         "input wire[1] ENB,";
attribute +         "input wire[1] WEB,";
attribute +         "input wire[1] SSRB,";
attribute +         "input wire[1] CLKB,";
attribute +         "output wire[4] DOB";

attribute +         "); \n";


//////////////////////////////////////////
// Component instantiation for the Xilinx//
// Block SelectRam with Dual Port        //
// A:2048 x 9 bits and B:2048 x 9 bits   //
//////////////////////////////////////////

attribute +     "void RAMB16_S9_S9(input wire[8] DIA,";
attribute +         "input wire[1] DIPA,";
attribute +         "input wire[11] ADDRA,";
attribute +         "input wire[1] ENA,";
attribute +         "input wire[1] WEA,";
attribute +         "input wire[1] SSRA,";
attribute +         "input wire[1] CLKA,";
attribute +         "output wire[8] DOA,";
attribute +         "output wire[1] DOPA,";

attribute +         "input wire[8] DIB,";
attribute +         "input wire[1] DIPB,";
attribute +         "input wire[11] ADDRB,";
attribute +         "input wire[1] ENB,";
attribute +         "input wire[1] WEB,";
attribute +         "input wire[1] SSRB,";
attribute +         "input wire[1] CLKB,";
attribute +         "output wire[8] DOB,";
```

```
        attribute +         "output wire[1] DOPB";

        attribute +         "); \n";

        ////////////////////////////////////////
        // Component instantiation for the Xilinx//
        // Block SelectRam with Dual Port        //
        // A:1024 x 18 bits and B:1024 x 18 bits //
        ////////////////////////////////////////

        attribute +     "void RAMB16_S18_S18(input wire[16] DIA,";
        attribute +         "input wire[2] DIPA,";
        attribute +         "input wire[10] ADDRA,";
        attribute +         "input wire[1] ENA,";
        attribute +         "input wire[1] WEA,";
        attribute +         "input wire[1] SSRA,";
        attribute +         "input wire[1] CLKA,";
        attribute +         "output wire[16] DOA,";
        attribute +         "output wire[2] DOPA,";

        attribute +         "input wire[16] DIB,";
        attribute +         "input wire[2] DIPB,";
        attribute +         "input wire[10] ADDRB,";
        attribute +         "input wire[1] ENB,";
        attribute +         "input wire[1] WEB,";
        attribute +         "input wire[1] SSRB,";
        attribute +         "input wire[1] CLKB,";
        attribute +         "output wire[16] DOB,";
        attribute +         "output wire[2] DOPB";

        attribute +         "); \n";

        attribute +     " void UNSIGNED_DIVIDER ( input wire[16]
dividend,";
        attribute +           "input wire[16] divisor,";
        attribute +           "output wire[16] quot,";
        attribute +           "output wire[16] remd,";
        attribute +           "input wire[1] c";
        attribute +           ");\n";

        attribute +     " void SIGNED_DIVIDER ( input wire[16]
dividend,";
        attribute +           "input wire[16] divisor,";
        attribute +           "output wire[16] quot,";
        attribute +           "output wire[16] remd,";
        attribute +           "input wire[1] c";
        attribute +           ");\n";

    }
}

////////////////////////////////////////////////////////////
// Unsigned Division Core IP                              ///
////////////////////////////////////////////////////////////
```

```
RESOURCEDEF UNSIGNED_DIVIDER
{
    FUNCTIONALITY DIV;
    ATTRIBUTE SOFTCORE
      {
           <DIV>
                {
                  attribute + "true";
                 }
       }
 ATTRIBUTE INSTANTIATION
      {
          <DIV>
           {
            int before_dec;
            int after_dec;
            before_dec = 0;
            after_dec = 0;

            after_dec = in1->fractional_bits();
            before_dec = 16 - after_dec;

            attribute +
                    " input wrap unsigned fixed [" + before_dec + "," +
after_dec + "] " + component_name + "_dividend;\n" +
                    " input wrap unsigned fixed [" + before_dec + "," +
after_dec + "] " + component_name + "_divisor;\n" +

                    " output wrap unsigned fixed [16,0] " +
component_name + "_quot;\n" +
                    " output wrap unsigned fixed [1,15] " +
component_name + "_remd;\n";
             attribute +
                  " unsigned wrap fixed [7,0] " + component_name +
"_latency;\n";

            attribute +
                   " instantiate UNSIGNED_DIVIDER : " +  component_name
+
                                "("        +                    "_dividend =
" + component_name + "_dividend," +
                                    " divisor = " + component_name +
"_divisor," +
                                    " quot = " + component_name + "_quot,"
+
                          " remd = " + component_name + "_remd," +
                          " c = " + clock_name +
            ");";

                        }
  }

ATTRIBUTE CAN_DO
```

```
 {
    <DIV>
         {
              if ( in1->bitwidth( )<17 && in2->bitwidth( )<17 && in2->bitwidth( )>2 &&
                                                  in1->is_unsigned( ) == true && in2->is_unsigned( ) == true )
                   {
                        attribute + "true";
                   }
              else
                   {
                        attribute + "false";
                   }
         }
    }

ATTRIBUTE PIPE_DELAY
  {
       <DIV>
            {
              attribute  + "1";
            }
  }

ATTRIBUTE COMBINATIONAL
  {
      <DIV>
         {
           attribute + "false";
         }
  }

ATTRIBUTE NUM_STATES
  {
     <DIV>
        {
          attribute + "2";
        }
  }

ATTRIBUTE INTERFACE
  {
    <DIV>
       {
          int fract;
          int start_position_quot;
          int end_position_remd;
          int mid_position_result;
          fract = out1->fractional_bits();
          start_position_quot = 15 - fract;
          end_position_remd = 16 -fract;
          mid_position_result = fract -1;
```

```
            attribute + "state1: {" ;
            attribute + component_name + "_dividend = " + in1->name( ) + "
; \n";
            attribute + component_name + "_divisor = " + in2->name( ) + "
; \n";
            attribute + component_name + "_latency = 0 ; \n";
            attribute + " goto state2 ; \n";
            attribute + "};\n";
            attribute + "state2: {";
            attribute + " if (" + component_name + "_latency " + " >20
)\n";
            attribute + "       {  " + out1->name( ) + "(15:" + fract + ")
= " +
component_name +  "_quot( " + start_position_quot +
                                                    ":0  ) ;\n";
            attribute + "          " + out1->name( ) + "( " +
mid_position_result + ":0 ) ="
                                  + component_name + "_remd ( 15:" +
end_position_remd +
" ) ;\n";
            attribute + "          goto NEXTSTATE; \n";
            attribute + "       } \n";
            attribute + " else \n";
            attribute + "       {     " + component_name + "_latency = " +
component_name + "_latency + 1 ;\n";
            attribute + "       } \n";
            attribute + "};\n";


        }
    }
}

////////////////////////////////////////////////////////////////////////
// Signed Division IP Core                                            //
////////////////////////////////////////////////////////////////////////

RESOURCEDEF SIGNED_DIVIDER
{
    FUNCTIONALITY DIV;
    ATTRIBUTE SOFTCORE
      {
           <DIV>
                {
                  attribute + "true";
                 }
       }
    ATTRIBUTE INSTANTIATION
      {   <DIV>
           {
            int before_dec;
            int after_dec;
            before_dec = 0;
            after_dec = 0;
            after_dec = in1->fractional_bits();
```

```
            before_dec = 16 - after_dec;
            attribute +
                    " input wrap signed fixed [" + before_dec + "," +
after_dec+"] " + component_name + "_dividend;\n" +
                    " input wrap signed fixed [" + before_dec + "," +
after_dec+"] " + component_name + "_divisor;\n" +
                    " output wrap signed fixed [16,0] " + component_name
+ "_quot;\n" +
                    " output wrap signed fixed [1,15] " + component_name
+ "_remd;\n";
             attribute +
                  " unsigned wrap fixed [7,0] " + component_name +
"_latency;\n";

            attribute +
                   " instantiate SIGNED_DIVIDER : " +  component_name +
                                "("        +                  " dividend =
" + component_name + "_dividend," +
                                     " divisor = " + component_name +
"_divisor," +
                                     " quot = " + component_name + "_quot,"
+
                        " remd = " + component_name + "_remd," +
                        " c = " + clock_name +
            ");";

                      }
      }

ATTRIBUTE CAN_DO
 {
    <DIV>
         {
              if ( in1->bitwidth( )<17 && in2->bitwidth( )<17 && in2-
>bitwidth( )>2 &&
                                          in1->is_signed( ) == true && in2-
>is_signed( ) == true )
                {

                    attribute + "true";
                }
              else
                 {
                    attribute + "false";
                 }
           }
    }

ATTRIBUTE PIPE_DELAY
  {
       <DIV>
           {
             attribute  + "1";
           }
```

```
    }

ATTRIBUTE COMBINATIONAL
  {
      <DIV>
         {
           attribute + "false";
         }
    }

ATTRIBUTE NUM_STATES
  {
     <DIV>
        {
           attribute + "2";
        }
  }

ATTRIBUTE INTERFACE
  {
    <DIV>
       {
          int fract;
          int start_position_quot;
          int end_position_remd;
          int mid_position_result;
          fract = out1->fractional_bits();
          start_position_quot = 15 - fract;
          end_position_remd = 15 -fract;
          mid_position_result = fract -1;

          attribute + "state1: {" ;
          attribute + component_name + "_dividend = " + in1->name( ) + "
; \n";
          attribute + component_name + "_divisor = " + in2->name( ) + "
; \n";
          attribute + component_name + "_latency = 0 ; \n";
          attribute + " goto state2 ; \n";
          attribute + "}\n";
          attribute + "state2: {";
          attribute + " if (" + component_name + "_latency " + " >34
)\n";
          attribute + "      {    if ( " + component_name + "_remd < 0
)\n";
          attribute + "                 {  " + out1->name( ) + "(15:" +
fract + ") = " +
component_name + "_quot  - 1 ;\n";
          attribute + "                 }   \n";
          attribute + "           else \n";
          attribute + "                 {  " + out1->name( ) + "(15:" +
fract + ") = " +
component_name +  "_quot;\n";
          attribute + "                 } \n";
```

```
            attribute + "          " + out1->name( ) + "( " +
mid_position_result + ":0 ) ="
                                    + component_name + "_remd ( 14:" +
end_position_remd +
" ) ;\n";
            attribute + "          goto NEXTSTATE; \n";
            attribute + "       }   \n";
            attribute + " else \n";
            attribute + "       {    " + component_name + "_latency = " +
component_name + "_latency + 1 ;\n";
            attribute + "       }  \n";
            attribute + "}\n";

        }
    }
}



////////////////////////////////////////////////////////////////////
// Functionality and Architecture description for the XC2V250     //
// Embedded Multipliers. There are 24 multipliers arranged in 4   //
// columns with 6 multipliers per column. The location properties //
// use the following form : LOC = MULT18X18_X#Y#                  //
////////////////////////////////////////////////////////////////////

RESOURCEDEF MULT18X18
{

        ////////////////////////////////////////
        // A[15:0] : Input "in1" to multiplier  //
        // B[15:0] : Input "in2" to multiplier  //
        // P[15:0] : Output "out1" to multiplier //
        ////////////////////////////////////////

        ////////////////////////////////////////
        // Embedded multipliers                //
        ////////////////////////////////////////

        FUNCTIONALITY MULT;

        ////////////////////////////////////////
        // This number of Embedded multiplier   //
        // resources that are used/instantiated  //
        // by the compiler can be controlled by  //
        // the user. This attribute informs the //
        // compiler that this is user           //
        // contollable resource.                //
        ////////////////////////////////////////

        ATTRIBUTE USERCONTROL
           {
                attribute + "true";
           }
```

```
                    ATTRIBUTE INSTANTIATION
                {
                  ////////////////////////////////////////////
                  // component_name is the specific Embedded  //
                  // Multiplier that needs to be instantiated //
                  ////////////////////////////////////////////

                      attribute +
                         "input signed fixed[18,0] " + component_name +
"_A;\n" +
                         "input signed fixed[18,0] " + component_name +
"_B;\n" +
                         "output signed fixed[36,0] " + component_name +
"_P;\n";

                      attribute +
                         "instantiate MULT18X18 : " + component_name +
                          "(" +
                                       "A = " + component_name + "_A," +
                                "B = " + component_name + "_B," +
                                "P = " + component_name + "_P" +
                               ");";
                }

             ////////////////////////////////////////
             // This attribute returns true if the    //
             // MULT18X18 can be used to perform the  //
             // multiplication operation on the inputs//
             // The embedded multiplier is used for   //
             // signed multiplication with the MSB as //
             // the sign bit. Hence, we can have a    //
             // maximum of 17X17 unsigned mults       //
             ////////////////////////////////////////

             ATTRIBUTE CAN_DO
                {
                   <MULT>
                      {
                      if(((in1->is_signed() == true && in1->bitwidth() < 19)
&&
                              (in2->is_signed() == true && in2->bitwidth() <
19)) ||
                             ((in1->is_unsigned() == true && in1->bitwidth()
< 18)&&
                              (in2->is_unsigned() == true && in2->bitwidth()
< 18)))
                        {
                        attribute + "true";
                        }
                      else
                       {
                        attribute + "false";
                           }
```

```
            }
        }

    ////////////////////////////////////////
    // The number of FSM states to wait for  //
    // before this same resource can be used //
    // for further computations.             //
    ////////////////////////////////////////

    ATTRIBUTE PIPE_DELAY
        {
            <MULT>
               {
                    attribute + "1";
               }
        }

    ////////////////////////////////////////
    // The embedded multipliers are supposed //
    // to be used as combinatorial resources //
    ////////////////////////////////////////

    ATTRIBUTE COMBINATIONAL
        {
            <MULT>
               {
                    attribute + "true";
               }
        }

    ////////////////////////////////////////
    // The number of FSM states that the     //
    // interface code spans. This is required//
    // so that the MIF Interface Parser      //
    // assigns the FSM State numbers properly//
    ////////////////////////////////////////

    ATTRIBUTE NUM_STATES
       {
          <MULT>
            {
                    attribute + "1";
            }
       }


    ////////////////////////////////////////
    // This resource can be used for input   //
    // bits less than 18. The interface code //
    // has to be generated based on the input//
    // bitwidths and the signtype            //
    ////////////////////////////////////////
```

```
ATTRIBUTE INTERFACE
   {
    <MULT>
        {
            int bitval1;
            int bitval2;
            int bitval;
            int outbitval;
            bitval1 = in1->bitwidth();
            bitval2 = in2->bitwidth();
            if(bitval1 < bitval2)
            {
               bitval = bitval2;
            }
            else
            {
               bitval = bitval1;
            }
            outbitval = 2 * bitval;

           attribute + "state1: {";
           if(bitval <= 4)
           {
                if(in1->is_signed() == true)
                {
                    attribute + component_name +"_A(17:8) =
0;\n";
                    attribute + component_name +"_A(7:0) = " +
                        in1->name() + ";\n";
                }
                else
                {
                    attribute + component_name +"_A(17:4) =
0;\n";
                    attribute + component_name +"_A(3:0) = " +
                        in1->name() + ";\n";
                }

                if(in2->is_signed() == true)
                {
                    attribute + component_name +"_B(17:8) =
0;\n";
                    attribute + component_name +"_B(7:0) = " +
                        in2->name() + ";\n";
                }
                else
                {
                    attribute + component_name +"_B(17:4) =
0;\n";
                    attribute + component_name +"_B(3:0) = " +
                        in2->name() + ";\n";
                }
           }
```

```
                    if(bitval <= 8 && bitval > 4)
                    {
                         if(in1->is_signed() == true)
                         {
                             attribute + component_name +"_A(17:16) =
    0;\n";
                             attribute + component_name +"_A(15:0) = " +
                                 in1->name() + ";\n";
                         }
                         else
                         {
                             attribute + component_name +"_A(17:8) =
    0;\n";
                             attribute + component_name +"_A(7:0) = " +
                                 in1->name() + ";\n";
                         }

                         if(in2->is_signed() == true)
                         {
                             attribute + component_name +"_B(17:16) =
    0;\n";
                             attribute + component_name +"_B(15:0) = " +
                                 in2->name() + ";\n";
                         }
                         else
                         {
                             attribute + component_name +"_B(17:8) =
    0;\n";
                             attribute + component_name +"_B(7:0) = " +
                                 in2->name() + ";\n";
                         }
                    }

                    if(bitval <= 18 && bitval > 8)
                    {
                         if(in1->is_signed() == true)
                         {
                             attribute + component_name +"_A(17:0) = " +
                                 in1->name() + ";\n";
                         }
                         else
                         {
                             attribute + component_name +"_A(17) = 0;\n";
                             attribute + component_name +"_A(16:0) = " +
                                 in1->name() + ";\n";
                         }

                         if(in2->is_signed() == true)
                         {
                             attribute + component_name +"_B(17:0) = " +
                                 in2->name() + ";\n";
                         }
                         else
                         {
```

```
                                attribute + component_name +"_B(17) = 0;\n";
                                attribute + component_name +"_B(16:0) = " +
                                    in2->name() + ";\n";
                            }
                        }
                        ////////////////////////////////////////
                        // Assign the output signal           //
                        ////////////////////////////////////////

                        outbitval = outbitval-1;
                        attribute + out1->name() + " = " +
                             component_name + "_P(" + outbitval + ": 0) ;
    \n";
                        attribute + "goto NEXTSTATE ;\n";
                        attribute + "}";
                }
                }
    }

    ////////////////////////////////////////////////////////////////
    // Functionality and Architecture description for the XC2V250 //
    // Embedded Synchronous Multipliers. These are RDL virtual    //
    // resources which use the MULT18X18                          //
    ////////////////////////////////////////////////////////////////

    RESOURCEDEF MULT18X18S
    {
            ////////////////////////////////////////
            // A[15:0] : Input "in1" to multiplier  //
            // B[15:0] : Input "in2" to multiplier  //
            // P[15:0] : Output "out1" to multiplier //
            ////////////////////////////////////////

            ////////////////////////////////////////
            // Embedded multipliers                //
            ////////////////////////////////////////

            FUNCTIONALITY MULT;

          ATTRIBUTE INSTANTIATION
               {
                 ////////////////////////////////////////////
                 // component_name is the specific Embedded  //
                 // Multiplier that needs to be instantiated //
                 ////////////////////////////////////////////

                    attribute +
                       "input signed fixed[18,0] " + component_name +
    "_A;\n" +
                       "input signed fixed[18,0] " + component_name +
    "_B;\n" +
```

```
                    "output signed fixed[36,0] " + component_name +
"_P;\n";

                attribute +
                   "instantiate MULT18X18S : " + component_name +
                       "(" +
                              "A = " + component_name + "_A," +
                              "B = " + component_name + "_B," +
                              "C = " + clock_name + "," +
                              "CE = " + "1," +
                              "R = " + reset_name + "," +
                              "P = " + component_name + "_P" +
                        ");";
            }

        ////////////////////////////////////////
        // This attribute returns true if the    //
        // MULT18X18S can be used to perform the //
        // multiplication operation on the inputs//
        // The embedded multiplier is used for   //
        // signed multiplication with the MSB as //
        // the sign bit. Hence, we can have a    //
        // maximum of 17X17 unsigned mults       //
        ////////////////////////////////////////

        ATTRIBUTE CAN_DO
           {
              <MULT>
                 {
                     if(((in1->is_signed() == true && in1->bitwidth() <
19) &&
                         (in2->is_signed() == true && in2->bitwidth() <
19)) ||
                        ((in1->is_unsigned() == true && in1->bitwidth()
< 18)&&
                         (in2->is_unsigned() == true && in2->bitwidth()
< 18)))
                      {
                         attribute + "true";
                      }
                     else
                      {
                         attribute + "false";
                      }
                 }
        }

        ////////////////////////////////////////
        // The number of FSM states to wait for  //
        // before this same resource can be used //
        // for further computations.             //
        ////////////////////////////////////////

        ATTRIBUTE PIPE_DELAY
```

```
    {
        <MULT>
           {
                attribute + "1";
           }
    }

////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
////////////////////////////////////////

ATTRIBUTE COMBINATIONAL
    {
        <MULT>
           {
                attribute + "false";
           }
    }

////////////////////////////////////////
// The number of FSM states that the     //
// interface code spans. This is required//
// so that the MIF Interface Parser      //
// assigns the FSM State numbers properly//
////////////////////////////////////////

ATTRIBUTE NUM_STATES
   {
      <MULT>
        {
                attribute + "2";
        }
   }


////////////////////////////////////////
// This resource can be used for input   //
// bits less than 18. The interface code //
// has to be generated based on the input//
// bitwidths and the signtype            //
////////////////////////////////////////


ATTRIBUTE INTERFACE
   {
    <MULT>
        {

            int bitval1;
            int bitval2;
            int bitval;
            int outbitval;
            bitval1 = in1->bitwidth();
```

```
                    bitval2 = in2->bitwidth();
                    if(bitval1 < bitval2)
                    {
                       bitval = bitval2;
                    }
                    else
                    {
                       bitval = bitval1;
                    }
                    outbitval = 2 * bitval;

                   attribute + "state1: {";
               if(bitval <= 18)
                   {
                        if(in1->is_signed() == true)
                        {
                            attribute + component_name +"_A(17:0) = " +
                                in1->name() + ";\n";
                        }
                        else
                        {
                            attribute + component_name +"_A(17) = 0;\n";
                            attribute + component_name +"_A(16:0) = " +
                                in1->name() + ";\n";
                        }

                        if(in2->is_signed() == true)
                        {
                            attribute + component_name +"_B(17:0) = " +
                                in2->name() + ";\n";
                        }
                        else
                        {
                            attribute + component_name +"_B(17) = 0;\n";
                            attribute + component_name +"_B(16:0) = " +
                                in2->name() + ";\n";
                        }
                    }
                    attribute + "goto state2 ;\n";
                    attribute + "}";
                    ////////////////////////////////////////////
                    // Assign the output signal               //
                    ////////////////////////////////////////////

                    attribute + "state2 : { \n";
                    outbitval = outbitval-1;
                    attribute + out1->name() + " = " +
                         component_name + "_P(" + outbitval + ": 0) ;
\n";
                    attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                }
            }
}
```

```
//////////////////////////////////////////////////////////////////////
// Functionality and Architecture description for the XC2V250 Block  //
// SelectRAM. This device has 24 SelectRAM memory blocks with storage //
// for 432 Kb of data. The location properties use the following     //
// form : LOC = RAMB16_X#Y#.                                         //
//////////////////////////////////////////////////////////////////////

FUNCTIONALITYDEF READ_MEM_ACCESS
{
   OUTPUT q;
}

FUNCTIONALITYDEF WRITE_MEM_ACCESS
{
   INPUT q;
}

//////////////////////////////////////////////////////////////////////
//  Read/Write Port A for true dual port memory                      //
//////////////////////////////////////////////////////////////////////

RESOURCEDEF RAM_PORT_A_S1_S1 {


        ////////////////////////////////////////
        // True Dual ported RAM               //
        ////////////////////////////////////////

        FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


        ////////////////////////////////////////
        // The number of FSM states to wait for  //
        // before this same resource can be used //
        // for further computations.          //
        ////////////////////////////////////////


        ATTRIBUTE PIPE_DELAY
           {
                attribute + "1";
           }

        ////////////////////////////////////////
        // The embedded multipliers are supposed //
        // to be used as combinatorial resources //
        ////////////////////////////////////////

        ATTRIBUTE COMBINATIONAL
           {
```

```
            <READ_MEM_ACCESS>
                 {
                     attribute + "false";
                 }
            <WRITE_MEM_ACCESS>
                 {
                     attribute + "false";
                 }
       }

//////////////////////////////////////////
// This attribute returns true if the   //
// input and output data widths are     //
// 1 bit wide                           //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
         <READ_MEM_ACCESS>
            {
                attribute + "true";
            }
         <WRITE_MEM_ACCESS>
            {
                attribute + "true";
            }
      }


//////////////////////////////////////////
// The number of FSM states that the    //
// interface code spans. This is required//
// so that the MIF Interface Parser     //
// assigns the FSM State numbers properly//
//////////////////////////////////////////

ATTRIBUTE NUM_STATES
   {
      <READ_MEM_ACCESS>
        {
                attribute + "3";
        }
      <WRITE_MEM_ACCESS>
        {
                attribute + "2";
        }
   }

//////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs            //
//////////////////////////////////////////

ATTRIBUTE INTERFACE
```

```
            {
                <READ_MEM_ACCESS>
                    {
                        int bitval;
                        bitval = q->bitwidth();
                        attribute + "state1 : {" +
                           component_name + "_WEA = 0; \n" +
                           component_name + "_ENA = 1; \n" +
                           component_name + "_DIA = 0; \n" +
                           component_name + "_DIPA = 0; \n" +
                           component_name + "_ADDRA = " +
                               mem_address + "; \n";

                         attribute + "goto state2 ;\n";
                       attribute + "}";

                        attribute + "state2 : {";
                        attribute + "goto state3 ;\n";
                       attribute + "}";

                        attribute + "state3 : {";
                        attribute + q->name() + "=" + component_name +
                                  "_DOA" + "; \n";
                        attribute + "goto NEXTSTATE ;\n";
                       attribute + "}";
                      }

                <WRITE_MEM_ACCESS>
                    {
                        int bitval;
                        bitval = q->bitwidth();
                        attribute + "state1 : {" +
                           component_name + "_WEA = 1; \n" +
                           component_name + "_ENA = 1; \n" +
                           component_name + "_ADDRA = " +
                               mem_address + "; \n";
                         attribute + "goto state2 ;\n";
                       attribute + "}";

                        attribute + "state2 : {";
                        attribute + component_name + "_DIA" + "=" +
                                    q->name() + "; \n";
                        attribute + "goto NEXTSTATE ;\n";
                       attribute + "}";
                      }
            }
    }

    ////////////////////////////////////////////////////////////////
    //  Read/Write Port A for true dual port memory               //
    ////////////////////////////////////////////////////////////////

    RESOURCEDEF RAM_PORT_A_S2_S2 {
```

```
//////////////////////////////////////////
// True Dual ported RAM                 //
//////////////////////////////////////////

FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


//////////////////////////////////////////
// The number of FSM states to wait for  //
// before this same resource can be used //
// for further computations.             //
//////////////////////////////////////////


ATTRIBUTE PIPE_DELAY
   {
        attribute + "1";
   }

//////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
//////////////////////////////////////////

ATTRIBUTE COMBINATIONAL
   {
       <READ_MEM_ACCESS>
            {
                attribute + "false";
            }
       <WRITE_MEM_ACCESS>
            {
                attribute + "false";
            }
   }

//////////////////////////////////////////
// This attribute returns true if the    //
// input and output data widths are      //
// 1 bit wide                            //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
         <READ_MEM_ACCESS>
            {
                attribute + "true";
            }
         <WRITE_MEM_ACCESS>
            {
                attribute + "true";
            }
      }
```

```
//////////////////////////////////////////
// The number of FSM states that the    //
// interface code spans. This is required//
// so that the MIF Interface Parser     //
// assigns the FSM State numbers properly//
//////////////////////////////////////////

ATTRIBUTE NUM_STATES
   {
      <READ_MEM_ACCESS>
        {
                attribute + "3";
        }
      <WRITE_MEM_ACCESS>
        {
                attribute + "2";
        }
   }


//////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs            //
//////////////////////////////////////////

ATTRIBUTE INTERFACE
   {
        <READ_MEM_ACCESS>
             {
                int bitval;
                bitval = q->bitwidth();
                attribute + "state1 : {" +
                   component_name + "_WEA = 0; \n" +
                   component_name + "_ENA = 1; \n" +
                   component_name + "_DIA = 0; \n" +
                   component_name + "_DIPA = 0; \n" +
                   component_name + "_ADDRA = " +
                       mem_address + "; \n";

                    attribute + "goto state2 ;\n";
               attribute + "}";

                    attribute + "state2 : {";
                    attribute + "goto state3 ;\n";
               attribute + "}";

                    attribute + "state3 : {";
                    attribute + q->name() + "=" + component_name +
                              "_DOA" + "; \n";
                    attribute + "goto NEXTSTATE ;\n";
               attribute + "}";
              }
```

```
                    <WRITE_MEM_ACCESS>
                         {
                             int bitval;
                             bitval = q->bitwidth();
                             attribute + "state1 : {" +
                                component_name + "_WEA = 1; \n" +
                                component_name + "_ENA = 1; \n" +
                                component_name + "_ADDRA = " +
                                    mem_address + "; \n";
                              attribute + "goto state2 ;\n";
                           attribute + "}";

                              attribute + "state2 : {";
                              attribute + component_name + "_DIA" + "=" +
                                          q->name() + "; \n";
                              attribute + "goto NEXTSTATE ;\n";
                           attribute + "}";
                          }
               }
     }

     ////////////////////////////////////////////////////////////////
     //  Read/Write Port A for true dual port memory                //
     ////////////////////////////////////////////////////////////////

     RESOURCEDEF RAM_PORT_A_S4_S4 {


          ////////////////////////////////////////
          // True Dual ported RAM               //
          ////////////////////////////////////////

          FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


          ////////////////////////////////////////
          // The number of FSM states to wait for  //
          // before this same resource can be used //
          // for further computations.          //
          ////////////////////////////////////////


          ATTRIBUTE PIPE_DELAY
             {
                  attribute + "1";
             }

          ////////////////////////////////////////
          // The embedded multipliers are supposed //
          // to be used as combinatorial resources //
          ////////////////////////////////////////

          ATTRIBUTE COMBINATIONAL
```

```
    {
        <READ_MEM_ACCESS>
            {
                attribute + "false";
            }
        <WRITE_MEM_ACCESS>
            {
                attribute + "false";
            }
    }

//////////////////////////////////////////
// This attribute returns true if the   //
// input and output data widths are     //
// 1 bit wide                           //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
         <READ_MEM_ACCESS>
            {
                attribute + "true";
            }
         <WRITE_MEM_ACCESS>
            {
                attribute + "true";
            }
      }


//////////////////////////////////////////
// The number of FSM states that the    //
// interface code spans. This is required//
// so that the MIF Interface Parser     //
// assigns the FSM State numbers properly//
//////////////////////////////////////////

ATTRIBUTE NUM_STATES
   {
      <READ_MEM_ACCESS>
        {
                attribute + "3";
        }
      <WRITE_MEM_ACCESS>
        {
                attribute + "2";
        }
   }

//////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs            //
//////////////////////////////////////////
```

```
ATTRIBUTE INTERFACE
   {
        <READ_MEM_ACCESS>
             {
                 int bitval;
                 bitval = q->bitwidth();
                 attribute + "state1 : {" +
                    component_name + "_WEA = 0; \n" +
                    component_name + "_ENA = 1; \n" +
                    component_name + "_DIA = 0; \n" +
                    component_name + "_DIPA = 0; \n" +
                    component_name + "_ADDRA = " +
                        mem_address + "; \n";

                  attribute + "goto state2 ;\n";
               attribute + "}";

                  attribute + "state2 : {";
                  attribute + "goto state3 ;\n";
               attribute + "}";

                  attribute + "state3 : {";
                  attribute + q->name() + "=" + component_name +
                              "_DOA" + "; \n";
                  attribute + "goto NEXTSTATE ;\n";
               attribute + "}";
              }

        <WRITE_MEM_ACCESS>
             {
                 int bitval;
                 bitval = q->bitwidth();
                 attribute + "state1 : {" +
                    component_name + "_WEA = 1; \n" +
                    component_name + "_ENA = 1; \n" +
                    component_name + "_ADDRA = " +
                        mem_address + "; \n";
                  attribute + "goto state2 ;\n";
               attribute + "}";

                  attribute + "state2 : {";
                  attribute + component_name + "_DIA" + "=" +
                              q->name() + "; \n";
                  attribute + "goto NEXTSTATE ;\n";
               attribute + "}";
              }
   }
}

////////////////////////////////////////////////////////////////
//  Read/Write Port A for true dual port memory               //
////////////////////////////////////////////////////////////////

RESOURCEDEF RAM_PORT_A_S9_S9 {
```

```
//////////////////////////////////////////
// True Dual ported RAM                 //
//////////////////////////////////////////

FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


//////////////////////////////////////////
// The number of FSM states to wait for  //
// before this same resource can be used //
// for further computations.             //
//////////////////////////////////////////

ATTRIBUTE PIPE_DELAY
   {
        attribute + "1";
   }

//////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
//////////////////////////////////////////

ATTRIBUTE COMBINATIONAL
   {
       <READ_MEM_ACCESS>
            {
                attribute + "false";
            }
       <WRITE_MEM_ACCESS>
            {
                attribute + "false";
            }
   }

//////////////////////////////////////////
// This attribute returns true if the    //
// input and output data widths are      //
// 1 bit wide                            //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
         <READ_MEM_ACCESS>
            {
                attribute + "true";
            }
         <WRITE_MEM_ACCESS>
            {
                attribute + "true";
            }
```

```
        }

        ////////////////////////////////////////
        // The number of FSM states that the  //
        // interface code spans. This is required//
        // so that the MIF Interface Parser   //
        // assigns the FSM State numbers properly//
        ////////////////////////////////////////

        ATTRIBUTE NUM_STATES
           {
              <READ_MEM_ACCESS>
                {
                        attribute + "3";
                }
              <WRITE_MEM_ACCESS>
                {
                        attribute + "2";
                }
           }

        ////////////////////////////////////////
        // The interface for reading and writing //
        // from the Block SelectRAMs          //
        ////////////////////////////////////////

        ATTRIBUTE INTERFACE
           {
                <READ_MEM_ACCESS>
                     {
                         int bitval;
                         bitval = q->bitwidth();
                         attribute + "state1 : {" +
                            component_name + "_WEA = 0; \n" +
                            component_name + "_ENA = 1; \n" +
                            component_name + "_DIA = 0; \n" +
                            component_name + "_DIPA = 0; \n" +
                            component_name + "_ADDRA = " +
                                mem_address + "; \n";

                          attribute + "goto state2 ;\n";
                      attribute + "}";

                          attribute + "state2 : {";
                          attribute + "goto state3 ;\n";
                      attribute + "}";

                          attribute + "state3 : {";

                           if(q->bitwidth() <= 8)
                               {
                                attribute + q->name() + "=" +
component_name +
```

```
                                        "_DOA" + "; \n";
                                    }
                                else
                                    {
                                     bitval = bitval - 1;
                                     attribute + q->name() + "(" + bitval +
":8" +
                                         component_name + "_DOPA" + "; \n";
                                     attribute + q->name() + "(7:0) = " +
                                         component_name + "_DOA" + "; \n";
                                    }
                                attribute + "goto NEXTSTATE ;\n";
                            attribute + "}";
                           }

                    <WRITE_MEM_ACCESS>
                          {
                              int bitval;
                              bitval = q->bitwidth();
                              attribute + "state1 : {" +
                                 component_name + "_WEA = 1; \n" +
                                 component_name + "_ENA = 1; \n" +
                                 component_name + "_ADDRA = " +
                                     mem_address + "; \n";
                               attribute + "goto state2 ;\n";
                            attribute + "}";

                              attribute + "state2 : {";

                              if(q->bitwidth() <= 8)
                                   {
                                    attribute + component_name + "_DIPA = 0;
\n";
                                    attribute + component_name + "_DIA" +
"=" +
                                          q->name() + "; \n";
                                   }
                               else
                                   {
                                    bitval = bitval - 1;
                                    attribute + component_name + "_DIPA = "
+
                                         q->name() + "(" + bitval + ")" + ";
\n";
                                    bitval = bitval - 1;
                                    attribute + component_name + "_DIA" +
"=" +
                                     q->name() + "(" + bitval +
                                     ":0)" + ";\n";
                                   }
                              attribute + "goto NEXTSTATE ;\n";
                            attribute + "}";
                           }
                }
```

```
}

////////////////////////////////////////////////////////////////
//  Read/Write Port A for true dual port memory                //
////////////////////////////////////////////////////////////////

RESOURCEDEF RAM_PORT_A_S18_S18 {

        ////////////////////////////////////////
        // True Dual ported RAM               //
        ////////////////////////////////////////

        FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

        ////////////////////////////////////////
        // The number of FSM states to wait for  //
        // before this same resource can be used //
        // for further computations.          //
        ////////////////////////////////////////


        ATTRIBUTE PIPE_DELAY
           {
                attribute + "1";
           }

        ////////////////////////////////////////
        // The embedded multipliers are supposed //
        // to be used as combinatorial resources //
        ////////////////////////////////////////

        ATTRIBUTE COMBINATIONAL
           {
               <READ_MEM_ACCESS>
                     {
                         attribute + "false";
                     }
               <WRITE_MEM_ACCESS>
                     {
                         attribute + "false";
                     }
           }

        ////////////////////////////////////////
        // This attribute returns true if the    //
        // input and output data widths are      //
        // 1 bit wide                            //
        ////////////////////////////////////////

        ATTRIBUTE CAN_DO
             {
                 <READ_MEM_ACCESS>
```

```
                {
                    attribute + "true";
                }
            <WRITE_MEM_ACCESS>
                {
                    attribute + "true";
                }
        }

//////////////////////////////////////////
// The number of FSM states that the    //
// interface code spans. This is required//
// so that the MIF Interface Parser     //
// assigns the FSM State numbers properly//
//////////////////////////////////////////

ATTRIBUTE NUM_STATES
    {
        <READ_MEM_ACCESS>
            {
                attribute + "3";
            }
        <WRITE_MEM_ACCESS>
            {
                attribute + "2";
            }
    }

//////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs            //
//////////////////////////////////////////

ATTRIBUTE INTERFACE
    {
        <READ_MEM_ACCESS>
            {
                int bitval;
                bitval = q->bitwidth();
                attribute + "state1 : {" +
                   component_name + "_WEA = 0; \n" +
                   component_name + "_ENA = 1; \n" +
                   component_name + "_DIA = 0; \n" +
                   component_name + "_DIPA = 0; \n" +
                   component_name + "_ADDRA = " +
                       mem_address + "; \n";

                 attribute + "goto state2 ;\n";
              attribute + "}";

                 attribute + "state2 : {";
                 attribute + "goto state3 ;\n";
              attribute + "}";
```

```
                    attribute + "state3 : {";
                    if(q->bitwidth() <= 16)
                         {
                          attribute + q->name() + "=" +
component_name +
                              "_DOA" + "; \n";
                         }
                    else
                         {
                          bitval = bitval - 1;
                          attribute + q->name() + "(" + bitval +
":16" +
                              component_name + "_DOPA" + "; \n";
                          attribute + q->name() + "(15:0) = " +
                              component_name + "_DOA" + "; \n";
                         }
                     attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
               }

          <WRITE_MEM_ACCESS>
               {
                   int bitval;
                   bitval = q->bitwidth();
                   attribute + "state1 : {" +
                      component_name + "_WEA = 1; \n" +
                      component_name + "_ENA = 1; \n" +
                      component_name + "_ADDRA = " +
                          mem_address + "; \n";
                    attribute + "goto state2 ;\n";
               attribute + "}";

                    attribute + "state2 : {";
                    if(q->bitwidth() <= 16)
                         {
                          attribute + component_name + "_DIPA = 0;
\n";
                          attribute + component_name + "_DIA" +
"=" +
                                q->name() + "; \n";
                         }
                    else
                         {
                          bitval = bitval - 1;
                          attribute + component_name + "_DIPA = "
+
                               q->name() + "(" + bitval + ")" + ";
\n";
                          bitval = bitval - 1;
                          attribute + component_name + "_DIA" +
"=" +
                                q->name() + "(" + bitval +
                                ":0)" + ";\n";
```

```
                                }
                            attribute + "goto NEXTSTATE ;\n";
                        attribute + "}";
                       }
            }
}

////////////////////////////////////////////////////////////////
//  Read/Write Port A for true dual port memory               //
////////////////////////////////////////////////////////////////

RESOURCEDEF RAM_PORT_A_S36_S36 {


        ////////////////////////////////////////
        // True Dual ported RAM               //
        ////////////////////////////////////////

        FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


        ////////////////////////////////////////
        // The number of FSM states to wait for  //
        // before this same resource can be used //
        // for further computations.          //
        ////////////////////////////////////////


        ATTRIBUTE PIPE_DELAY
           {
                attribute + "1";
           }

        ////////////////////////////////////////
        // The embedded multipliers are supposed //
        // to be used as combinatorial resources //
        ////////////////////////////////////////

        ATTRIBUTE COMBINATIONAL
           {
               <READ_MEM_ACCESS>
                    {
                        attribute + "false";
                    }
               <WRITE_MEM_ACCESS>
                    {
                        attribute + "false";
                    }
           }

        ////////////////////////////////////////
        // This attribute returns true if the    //
        // input and output data widths are      //
        // 1 bit wide                            //
```

```
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
          <READ_MEM_ACCESS>
             {
                 attribute + "true";
             }
          <WRITE_MEM_ACCESS>
             {
                 attribute + "true";
             }
       }


//////////////////////////////////////////
// The number of FSM states that the     //
// interface code spans. This is required//
// so that the MIF Interface Parser      //
// assigns the FSM State numbers properly//
//////////////////////////////////////////

ATTRIBUTE NUM_STATES
   {
      <READ_MEM_ACCESS>
        {
                attribute + "3";
        }
      <WRITE_MEM_ACCESS>
        {
                attribute + "2";
        }
   }

//////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs             //
//////////////////////////////////////////

ATTRIBUTE INTERFACE
   {
        <READ_MEM_ACCESS>
             {
                 int bitval;
                 bitval = q->bitwidth();
                 attribute + "state1 : {" +
                    component_name + "_WEA = 0; \n" +
                    component_name + "_ENA = 1; \n" +
                    component_name + "_DIA = 0; \n" +
                    component_name + "_DIPA = 0; \n" +
                    component_name + "_ADDRA = " +
                        mem_address + "; \n";

                 attribute + "goto state2 ;\n";
```

```
                    attribute + "}";

                        attribute + "state2 : {";
                        attribute + "goto state3 ;\n";
                    attribute + "}";

                        attribute + "state3 : {";
                        if(q->bitwidth() <= 32)
                             {
                              attribute + q->name() + "=" +
component_name +
                                   "_DOA" + "; \n";
                             }
                        else
                             {
                              bitval = bitval - 1;
                              attribute + q->name() + "(" + bitval +
":32" +
                                   component_name + "_DOPA" + "; \n";
                              attribute + q->name() + "(31:0) = " +
                                   component_name + "_DOA" + "; \n";
                             }
                         attribute + "goto NEXTSTATE ;\n";
                     attribute + "}";
                    }

               <WRITE_MEM_ACCESS>
                    {
                        int bitval;
                        bitval = q->bitwidth();
                        attribute + "state1 : {" +
                           component_name + "_WEA = 1; \n" +
                           component_name + "_ENA = 1; \n" +
                           component_name + "_ADDRA = " +
                               mem_address + "; \n";
                         attribute + "goto state2 ;\n";
                     attribute + "}";

                         attribute + "state2 : {";
                         if(q->bitwidth() <= 32)
                              {
                               attribute + component_name + "_DIPA = 0;
\n";
                               attribute + component_name + "_DIA" +
"=" +
                                     q->name() + "; \n";
                              }
                         else
                              {
                               bitval = bitval - 1;
                               attribute + component_name + "_DIPA = "
+
                                     q->name() + "(" + bitval + ")" + ";
\n";
```

```
                                    bitval = bitval - 1;
                                    attribute + component_name + "_DIA" +
"=" +
                                          q->name() + "(" + bitval +
                                          ":0)" + ";\n";
                                }
                            attribute + "goto NEXTSTATE ;\n";
                        attribute + "}";
                        }
            }
}


////////////////////////////////////////////////////////////////
//  Read/Write Port B for true dual port memory                //
////////////////////////////////////////////////////////////////

RESOURCEDEF RAM_PORT_B_S1_S1 {

        ////////////////////////////////////////
        // True Dual ported RAM               //
        ////////////////////////////////////////

        FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


        ////////////////////////////////////////
        // The number of FSM states to wait for  //
        // before this same resource can be used //
        // for further computations.          //
        ////////////////////////////////////////

        ATTRIBUTE PIPE_DELAY
           {
                attribute + "1";
           }

        ////////////////////////////////////////
        // The embedded multipliers are supposed //
        // to be used as combinatorial resources //
        ////////////////////////////////////////

        ATTRIBUTE COMBINATIONAL
           {
               <READ_MEM_ACCESS>
                    {
                        attribute + "false";
                    }
               <WRITE_MEM_ACCESS>
                    {
```

```
                    attribute + "false";
                }
        }

    ////////////////////////////////////////
    // This attribute returns true if the  //
    // input and output data widths are    //
    // 1 bit wide                          //
    ////////////////////////////////////////

    ATTRIBUTE CAN_DO
        {
            <READ_MEM_ACCESS>
                {
                    attribute + "true";
                }
            <WRITE_MEM_ACCESS>
                {
                    attribute + "true";
                }
        }


    ////////////////////////////////////////
    // The number of FSM states that the   //
    // interface code spans. This is required//
    // so that the MIF Interface Parser    //
    // assigns the FSM State numbers properly//
    ////////////////////////////////////////

    ATTRIBUTE NUM_STATES
        {
            <READ_MEM_ACCESS>
                {
                    attribute + "3";
                }
            <WRITE_MEM_ACCESS>
                {
                    attribute + "2";
                }
        }

    ////////////////////////////////////////
    // The interface for reading and writing //
    // from the Block SelectRAMs           //
    ////////////////////////////////////////

    ATTRIBUTE INTERFACE
        {
            <READ_MEM_ACCESS>
                {
                    int bitval;
                    bitval = q->bitwidth();
                    attribute + "state1 : {" +
```

```
                        component_name + "_WEB = 0; \n" +
                        component_name + "_ENB = 1; \n" +
                        component_name + "_DIB = 0; \n" +
                        component_name + "_DIPB = 0; \n" +
                        component_name + "_ADDRB = " +
                            mem_address + "; \n";

                    attribute + "goto state2 ;\n";
                attribute + "}";

                    attribute + "state2 : {";
                    attribute + "goto state3 ;\n";
                attribute + "}";

                    attribute + "state3 : {";
                    attribute + q->name() + "=" + component_name +
                                "_DOB" + "; \n";
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                }

            <WRITE_MEM_ACCESS>
                {
                    int bitval;
                    bitval = q->bitwidth();
                    attribute + "state1 : {" +
                       component_name + "_WEB = 1; \n" +
                       component_name + "_ENB = 1; \n" +
                       component_name + "_ADDRB = " +
                           mem_address + "; \n";
                     attribute + "goto state2 ;\n";
                 attribute + "}";

                     attribute + "state2 : {";
                     attribute + component_name + "_DIB" + "=" +
                                 q->name() + "; \n";
                     attribute + "goto NEXTSTATE ;\n";
                 attribute + "}";
                }
        }
}

////////////////////////////////////////////////////////////////////
//  Read/Write Port B for true dual port memory                   //
////////////////////////////////////////////////////////////////////

RESOURCEDEF RAM_PORT_B_S2_S2 {

    ////////////////////////////////////////
    // True Dual ported RAM               //
    ////////////////////////////////////////

    FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;
```

```
//////////////////////////////////////////
// The number of FSM states to wait for  //
// before this same resource can be used //
// for further computations.             //
//////////////////////////////////////////


ATTRIBUTE PIPE_DELAY
   {
        attribute + "1";
   }

//////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
//////////////////////////////////////////

ATTRIBUTE COMBINATIONAL
   {
       <READ_MEM_ACCESS>
            {
                attribute + "false";
            }
       <WRITE_MEM_ACCESS>
            {
                attribute + "false";
            }
   }

//////////////////////////////////////////
// This attribute returns true if the    //
// input and output data widths are      //
// 1 bit wide                            //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
         <READ_MEM_ACCESS>
            {
                attribute + "true";
            }
         <WRITE_MEM_ACCESS>
            {
                attribute + "true";
            }
      }


//////////////////////////////////////////
// The number of FSM states that the     //
// interface code spans. This is required//
// so that the MIF Interface Parser      //
```

```
// assigns the FSM State numbers properly//
////////////////////////////////////////////

ATTRIBUTE NUM_STATES
   {
      <READ_MEM_ACCESS>
        {
                attribute + "3";
        }
      <WRITE_MEM_ACCESS>
        {
                attribute + "2";
        }
   }


////////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs             //
////////////////////////////////////////////

ATTRIBUTE INTERFACE
   {
        <READ_MEM_ACCESS>
             {
                 int bitval;
                 bitval = q->bitwidth();
                 attribute + "state1 : {" +
                    component_name + "_WEB = 0; \n" +
                    component_name + "_ENB = 1; \n" +
                    component_name + "_DIB = 0; \n" +
                    component_name + "_DIPB = 0; \n" +
                    component_name + "_ADDRB = " +
                        mem_address + "; \n";

                  attribute + "goto state2 ;\n";
               attribute + "}";

                  attribute + "state2 : {";
                  attribute + "goto state3 ;\n";
               attribute + "}";

                  attribute + "state3 : {";
                  attribute + q->name() + "=" + component_name +
                            "_DOB" + "; \n";
                  attribute + "goto NEXTSTATE ;\n";
               attribute + "}";
             }

        <WRITE_MEM_ACCESS>
             {
                 int bitval;
                 bitval = q->bitwidth();
                 attribute + "state1 : {" +
```

```
                                component_name + "_WEB = 1; \n" +
                                component_name + "_ENB = 1; \n" +
                                component_name + "_ADDRB = " +
                                    mem_address + "; \n";
                            attribute + "goto state2 ;\n";
                        attribute + "}";

                            attribute + "state2 : {";
                            attribute + component_name + "_DIB" + "=" +
                                        q->name() + "; \n";
                            attribute + "goto NEXTSTATE ;\n";
                        attribute + "}";
                        }
                }
}

////////////////////////////////////////////////////////////////
//  Read/Write Port B for true dual port memory                //
////////////////////////////////////////////////////////////////

RESOURCEDEF RAM_PORT_B_S4_S4 {


        ////////////////////////////////////////
        // True Dual ported RAM               //
        ////////////////////////////////////////

        FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


        ////////////////////////////////////////
        // The number of FSM states to wait for  //
        // before this same resource can be used //
        // for further computations.          //
        ////////////////////////////////////////


        ATTRIBUTE PIPE_DELAY
           {
                attribute + "1";
           }

        ////////////////////////////////////////
        // The embedded multipliers are supposed //
        // to be used as combinatorial resources //
        ////////////////////////////////////////

        ATTRIBUTE COMBINATIONAL
           {
               <READ_MEM_ACCESS>
                    {
                        attribute + "false";
                    }
               <WRITE_MEM_ACCESS>
```

```
                {
                    attribute + "false";
                }
        }

//////////////////////////////////////////
// This attribute returns true if the   //
// input and output data widths are     //
// 1 bit wide                           //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
         <READ_MEM_ACCESS>
            {
                attribute + "true";
            }
         <WRITE_MEM_ACCESS>
            {
                attribute + "true";
            }
      }


//////////////////////////////////////////
// The number of FSM states that the    //
// interface code spans. This is required//
// so that the MIF Interface Parser     //
// assigns the FSM State numbers properly//
//////////////////////////////////////////

ATTRIBUTE NUM_STATES
   {
      <READ_MEM_ACCESS>
        {
                attribute + "3";
        }
      <WRITE_MEM_ACCESS>
        {
                attribute + "2";
        }
   }

//////////////////////////////////////////
// The interface for reading and writing //
// from the Block SelectRAMs            //
//////////////////////////////////////////

ATTRIBUTE INTERFACE
   {
        <READ_MEM_ACCESS>
             {
                 int bitval;
                 bitval = q->bitwidth();
```

```
                        attribute + "state1 : {" +
                           component_name + "_WEB = 0; \n" +
                           component_name + "_ENB = 1; \n" +
                           component_name + "_DIB = 0; \n" +
                           component_name + "_DIPB = 0; \n" +
                           component_name + "_ADDRB = " +
                               mem_address + "; \n";

                        attribute + "goto state2 ;\n";
                    attribute + "}";

                        attribute + "state2 : {";
                        attribute + "goto state3 ;\n";
                    attribute + "}";

                        attribute + "state3 : {";
                        attribute + q->name() + "=" + component_name +
                                    "_DOB" + "; \n";
                        attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                    }

                <WRITE_MEM_ACCESS>
                    {
                        int bitval;
                        bitval = q->bitwidth();
                        attribute + "state1 : {" +
                           component_name + "_WEB = 1; \n" +
                           component_name + "_ENB = 1; \n" +
                           component_name + "_ADDRB = " +
                               mem_address + "; \n";
                        attribute + "goto state2 ;\n";
                    attribute + "}";

                        attribute + "state2 : {";
                        attribute + component_name + "_DIB" + "=" +
                                    q->name() + "; \n";
                        attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                    }
            }
    }

    ////////////////////////////////////////////////////////////////
    //  Read/Write Port B for true dual port memory                //
    ////////////////////////////////////////////////////////////////

    RESOURCEDEF RAM_PORT_B_S9_S9 {


        ////////////////////////////////////////
        // True Dual ported RAM                //
        ////////////////////////////////////////
```

```
FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


//////////////////////////////////////////
// The number of FSM states to wait for  //
// before this same resource can be used //
// for further computations.             //
//////////////////////////////////////////


ATTRIBUTE PIPE_DELAY
   {
        attribute + "1";
   }

//////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
//////////////////////////////////////////

ATTRIBUTE COMBINATIONAL
   {
       <READ_MEM_ACCESS>
            {
                attribute + "false";
            }
       <WRITE_MEM_ACCESS>
            {
                attribute + "false";
            }
   }

//////////////////////////////////////////
// This attribute returns true if the    //
// input and output data widths are      //
// 1 bit wide                            //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
         <READ_MEM_ACCESS>
            {
                attribute + "true";
            }
         <WRITE_MEM_ACCESS>
            {
                attribute + "true";
            }
       }


ATTRIBUTE NUM_STATES
```

```
            {
               <READ_MEM_ACCESS>
                 {
                        attribute + "3";
                 }
               <WRITE_MEM_ACCESS>
                 {
                        attribute + "2";
                 }
            }

         ////////////////////////////////////////
         // The interface for reading and writing //
         // from the Block SelectRAMs             //
         ////////////////////////////////////////

         ATTRIBUTE INTERFACE
            {
                 <READ_MEM_ACCESS>
                      {
                          int bitval;
                          bitval = q->bitwidth();
                          attribute + "state1 : {" +
                             component_name + "_WEB = 0; \n" +
                             component_name + "_ENB = 1; \n" +
                             component_name + "_DIB = 0; \n" +
                             component_name + "_DIPB = 0; \n" +
                             component_name + "_ADDRB = " +
                                 mem_address + "; \n";

                          attribute + "goto state2 ;\n";
                        attribute + "}";

                          attribute + "state2 : {";
                          attribute + "goto state3 ;\n";
                        attribute + "}";

                          attribute + "state3 : {";

                           if(q->bitwidth() <= 8)
                               {
                                attribute + q->name() + "=" +
component_name +
                                     "_DOB" + "; \n";
                               }
                           else
                               {
                                bitval = bitval - 1;
                                attribute + q->name() + "(" + bitval +
":8" +
                                     component_name + "_DOPB" + "; \n";
                                attribute + q->name() + "(7:0) = " +
                                     component_name + "_DOB" + "; \n";
                               }
```

```
                        attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                   }

5                <WRITE_MEM_ACCESS>
                    {
                       int bitval;
                       bitval = q->bitwidth();
                       attribute + "state1 : {" +
10                        component_name + "_WEB = 1; \n" +
                          component_name + "_ENB = 1; \n" +
                          component_name + "_ADDRB = " +
                              mem_address + "; \n";
                        attribute + "goto state2 ;\n";
15                    attribute + "}";

                        attribute + "state2 : {";

                        if(q->bitwidth() <= 8)
20                          {
                            attribute + component_name + "_DIPB = 0;
    \n";
                            attribute + component_name + "_DIB" +
    "=" +
25                                q->name() + "; \n";
                          }
                       else
                          {
                            bitval = bitval - 1;
30                          attribute + component_name + "_DIPB = "
    +
                                q->name() + "(" + bitval + ")" + ";
    \n";
                            bitval = bitval - 1;
35                          attribute + component_name + "_DIB" +
    "=" +
                                q->name() + "(" + bitval +
                                ":0)" + ";\n";
                          }
40                     attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                   }
            }
    }
45
    ////////////////////////////////////////////////////////////////
    //  Read/Write Port B for true dual port memory                //
    ////////////////////////////////////////////////////////////////

50  RESOURCEDEF RAM_PORT_B_S18_S18 {


        ////////////////////////////////////////
        // True Dual ported RAM                //
```

```
//////////////////////////////////////////

FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;


//////////////////////////////////////////
// The number of FSM states to wait for  //
// before this same resource can be used //
// for further computations.             //
//////////////////////////////////////////


ATTRIBUTE PIPE_DELAY
   {
        attribute + "1";
   }

//////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
//////////////////////////////////////////

ATTRIBUTE COMBINATIONAL
   {
       <READ_MEM_ACCESS>
            {
                attribute + "false";
            }
       <WRITE_MEM_ACCESS>
            {
                attribute + "false";
            }
   }

//////////////////////////////////////////
// This attribute returns true if the    //
// input and output data widths are      //
// 1 bit wide                            //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
     {
         <READ_MEM_ACCESS>
            {
                attribute + "true";
            }
         <WRITE_MEM_ACCESS>
            {
                attribute + "true";
            }
       }


ATTRIBUTE NUM_STATES
```

```
            {
               <READ_MEM_ACCESS>
                 {
                        attribute + "3";
                 }
               <WRITE_MEM_ACCESS>
                 {
                        attribute + "2";
                 }
            }

          ////////////////////////////////////////
          // The interface for reading and writing //
          // from the Block SelectRAMs             //
          ////////////////////////////////////////

          ATTRIBUTE INTERFACE
             {
                  <READ_MEM_ACCESS>
                       {
                           int bitval;
                           bitval = q->bitwidth();
                           attribute + "state1 : {" +
                              component_name + "_WEB = 0; \n" +
                              component_name + "_ENB = 1; \n" +
                              component_name + "_DIB = 0; \n" +
                              component_name + "_DIPB = 0; \n" +
                              component_name + "_ADDRB = " +
                                  mem_address + "; \n";

                           attribute + "goto state2 ;\n";
                        attribute + "}";

                           attribute + "state2 : {";
                           attribute + "goto state3 ;\n";
                        attribute + "}";

                           attribute + "state3 : {";
                           if(q->bitwidth() <= 16)
                                {
                                 attribute + q->name() + "=" +
component_name +
                                      "_DOB" + "; \n";
                                }
                           else
                                {
                                 bitval = bitval - 1;
                                 attribute + q->name() + "(" + bitval +
":16" +
                                      component_name + "_DOPB" + "; \n";
                                 attribute + q->name() + "(15:0) = " +
                                      component_name + "_DOB" + "; \n";
                                }
                           attribute + "goto NEXTSTATE ;\n";
```

```
                        attribute + "}";
                       }

                  <WRITE_MEM_ACCESS>
                       {
                           int bitval;
                           bitval = q->bitwidth();
                           attribute + "state1 : {" +
                              component_name + "_WEB = 1; \n" +
                              component_name + "_ENB = 1; \n" +
                              component_name + "_ADDRB = " +
                                  mem_address + "; \n";
                            attribute + "goto state2 ;\n";
                          attribute + "}";

                           attribute + "state2 : {";
                           if(q->bitwidth() <= 16)
                                {
                                 attribute + component_name + "_DIPB = 0;
\n";
                                 attribute + component_name + "_DIB" +
"=" +
                                       q->name() + "; \n";
                                }
                           else
                                {
                                 bitval = bitval - 1;
                                 attribute + component_name + "_DIPB = "
+
                                      q->name() + "(" + bitval + ")" + ";
\n";
                                 bitval = bitval - 1;
                                 attribute + component_name + "_DIB" +
"=" +
                                       q->name() + "(" + bitval +
                                       ":0)" + ";\n";
                                }
                           attribute + "goto NEXTSTATE ;\n";
                          attribute + "}";
                       }
             }
}

////////////////////////////////////////////////////////////////////
//  Read/Write Port B for true dual port memory                   //
////////////////////////////////////////////////////////////////////

RESOURCEDEF RAM_PORT_B_S36_S36 {


        ////////////////////////////////////////
        // True Dual ported RAM               //
        ////////////////////////////////////////
```

```
FUNCTIONALITY READ_MEM_ACCESS,WRITE_MEM_ACCESS;

//////////////////////////////////////////
// The number of FSM states to wait for  //
// before this same resource can be used //
// for further computations.             //
//////////////////////////////////////////

ATTRIBUTE PIPE_DELAY
   {
        attribute + "1";
   }

//////////////////////////////////////////
// The embedded multipliers are supposed //
// to be used as combinatorial resources //
//////////////////////////////////////////

ATTRIBUTE COMBINATIONAL
   {
       <READ_MEM_ACCESS>
            {
                attribute + "false";
            }
       <WRITE_MEM_ACCESS>
            {
                attribute + "false";
            }
   }

//////////////////////////////////////////
// This attribute returns true if the    //
// input and output data widths are      //
// 1 bit wide                            //
//////////////////////////////////////////

ATTRIBUTE CAN_DO
    {
        <READ_MEM_ACCESS>
           {
               attribute + "true";
           }
        <WRITE_MEM_ACCESS>
           {
               attribute + "true";
           }
     }

//////////////////////////////////////////
// The number of FSM states that the     //
// interface code spans. This is required//
```

```
            // so that the MIF Interface Parser      //
            // assigns the FSM State numbers properly//
            ////////////////////////////////////////////

            ATTRIBUTE NUM_STATES
               {
                  <READ_MEM_ACCESS>
                    {
                            attribute + "3";
                    }
                  <WRITE_MEM_ACCESS>
                    {
                            attribute + "2";
                    }
               }

            ////////////////////////////////////////////
            // The interface for reading and writing //
            // from the Block SelectRAMs             //
            ////////////////////////////////////////////

            ATTRIBUTE INTERFACE
               {
                    <READ_MEM_ACCESS>
                         {
                             int bitval;
                             bitval = q->bitwidth();
                             attribute + "state1 : {" +
                                component_name + "_WEB = 0; \n" +
                                component_name + "_ENB = 1; \n" +
                                component_name + "_DIB = 0; \n" +
                                component_name + "_DIPB = 0; \n" +
                                component_name + "_ADDRB = " +
                                    mem_address + "; \n";

                              attribute + "goto state2 ;\n";
                           attribute + "}";

                              attribute + "state2 : {";
                              attribute + "goto state3 ;\n";
                           attribute + "}";

                              attribute + "state3 : {";
                              if(q->bitwidth() <= 32)
                                   {
                                    attribute + q->name() + "=" +
component_name +
                                         "_DOB" + "; \n";
                                   }
                              else
                                   {
                                    bitval = bitval - 1;
                                    attribute + q->name() + "(" + bitval +
":32" +
```

```
                                        component_name + "_DOPB" + "; \n";
                                    attribute + q->name() + "(31:0) = " +
                                        component_name + "_DOB" + "; \n";
                                    }
                                attribute + "goto NEXTSTATE ;\n";
                            attribute + "}";
                            }

                        <WRITE_MEM_ACCESS>
                            {
                                int bitval;
                                bitval = q->bitwidth();
                                attribute + "state1 : {" +
                                   component_name + "_WEB = 1; \n" +
                                   component_name + "_ENB = 1; \n" +
                                   component_name + "_ADDRB = " +
                                       mem_address + "; \n";
                                attribute + "goto state2 ;\n";
                            attribute + "}";

                                attribute + "state2 : {";
                                if(q->bitwidth() <= 32)
                                    {
                                    attribute + component_name + "_DIPB = 0;
\n";
                                    attribute + component_name + "_DIB" +
"=" +
                                          q->name() + "; \n";
                                    }
                                else
                                    {
                                    bitval = bitval - 1;
                                    attribute + component_name + "_DIPB = "
+
                                         q->name() + "(" + bitval + ")" + ";
\n";
                                    bitval = bitval - 1;
                                    attribute + component_name + "_DIB" +
"=" +
                                          q->name() + "(" + bitval +
                                          ":0)" + ";\n";
                                    }
                                attribute + "goto NEXTSTATE ;\n";
                            attribute + "}";
                            }
                }
}


////////////////////////////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S1_S1 with aspect ratio of both ports as      //
// A: 16,384 x 1 bit and B: 16,384 X 1 bit               //
```

```
////////////////////////////////////////////////////////

RESOURCEDEF RAM_MODE_S1_S1 {

    ////////////////////////////////
    // This is connected to Ports //
    // which has read/write func  //
    ////////////////////////////////

    FUNCTIONALITY STORAGE;

    ////////////////////////////
    // A Dual Port RAM        //
    ////////////////////////////

    TAGS DUALPORT;

    ////////////////////////////////////////
    // The number of addressable locations //
    ////////////////////////////////////////

    ATTRIBUTE MAX_LOCATIONS
       {
            attribute + 16384;
       }

    ////////////////////////////////////////////
    // The width of each addressable locations //
    ////////////////////////////////////////////

    ATTRIBUTE MAX_WIDTH
       {
            attribute + 1;
       }

    ////////////////////////////////////////
    // The code to instantiate the RAM    //
    // in mode S1_S1                      //
    ////////////////////////////////////////

    ATTRIBUTE INSTANTIATION
       {
         ////////////////////////////////////////////
         // component_name is the specific RAM that //
         // needs to be accessed                    //
         ////////////////////////////////////////////


         ////////////////////////////////////////////
         // These signals have to be declared so that//
         // the compiler has them in the Symbol Table//
         // before it parses in the MIF interface    //
         // code from the ports.                     //
         ////////////////////////////////////////////
```

```
        attribute +
                    "input unsigned fixed[14,0] " + component_name +
"_ADDRA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_DIA;\n"+
                    "output unsigned fixed[1,0] " + component_name +
"_DOA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENA;\n";


        attribute +
                    "input unsigned fixed[14,0] " + component_name +
"_ADDRB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_DIB;\n"+
                    "output unsigned fixed[1,0] " + component_name +
"_DOB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENB;\n";

         attribute +
            "instantiate RAMB16_S1_S1 : " + component_name +
            "(" +          "DIA = " + component_name + "_DIA," +
                         "ADDRA = " + component_name + "_ADDRA," +
                         "DOA = " + component_name + "_DOA," +
                         "ENA = " + component_name + "_ENA," +
                         "WEA = " + component_name + "_WEA," +
                               "CLKA = " + clock_name + " , " +
                               "SSRA = " + reset_name + " , " +
                           "DIB = " + component_name + "_DIB," +
                         "ADDRB = " + component_name + "_ADDRB," +
                         "DOB = " + component_name + "_DOB," +
                         "ENB = " + component_name + "_ENB," +
                         "WEB = " + component_name + "_WEB," +
                               "CLKB = " + clock_name + " , " +
                               "SSRB = " + reset_name +
                          ");";

          }
}


////////////////////////////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S2_S2 with aspect ratio of both ports as      //
// A: 8192 x 2 bit and B: 8192 X 2 bit                   //
////////////////////////////////////////////////////////////
```

```
RESOURCEDEF RAM_MODE_S2_S2 {

        ////////////////////////////////
        // This is connected to Ports //
        // which has read/write func  //
        ////////////////////////////////

        FUNCTIONALITY STORAGE;

        ////////////////////////////
        // A Dual Port RAM        //
        ////////////////////////////

        TAGS DUALPORT;

        ////////////////////////////////////////
        // The number of addressable locations //
        ////////////////////////////////////////

        ATTRIBUTE MAX_LOCATIONS
           {
                attribute + 8192;
           }

        ////////////////////////////////////////////
        // The width of each addressable locations //
        ////////////////////////////////////////////

        ATTRIBUTE MAX_WIDTH
           {
                attribute + 2;
           }

        ////////////////////////////////////////
        // The code to instantiate the RAM    //
        // in mode S2_S2                      //
        ////////////////////////////////////////

        ATTRIBUTE INSTANTIATION
           {
             ////////////////////////////////////////////
             // component_name is the specific RAM that //
             // needs to be accessed                    //
             ////////////////////////////////////////////


             ////////////////////////////////////////////
             // These signals have to be declared so that//
             // the compiler has them in the Symbol Table//
             // before it parses in the MIF interface    //
             // code from the ports.                     //
             ////////////////////////////////////////////
```

```
        attribute +
                    "input unsigned fixed[13,0] " + component_name +
"_ADDRA;\n"+
                    "input unsigned fixed[2,0] " + component_name +
"_DIA;\n"+
                    "output unsigned fixed[2,0] " + component_name +
"_DOA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENA;\n";

        attribute +
                    "input unsigned fixed[13,0] " + component_name +
"_ADDRB;\n"+
                    "input unsigned fixed[2,0] " + component_name +
"_DIB;\n"+
                    "output unsigned fixed[2,0] " + component_name +
"_DOB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENB;\n";

         attribute +
            "instantiate RAMB16_S2_S2 : " + component_name +
            "(" +          "DIA = " + component_name + "_DIA," +
                         "ADDRA = " + component_name + "_ADDRA," +
                         "DOA = " + component_name + "_DOA," +
                         "ENA = " + component_name + "_ENA," +
                         "WEA = " + component_name + "_WEA," +
                               "CLKA = " + clock_name + " , " +
                               "SSRA = " + reset_name + " , " +
                           "DIB = " + component_name + "_DIB," +
                         "ADDRB = " + component_name + "_ADDRB," +
                         "DOB = " + component_name + "_DOB," +
                         "ENB = " + component_name + "_ENB," +
                         "WEB = " + component_name + "_WEB," +
                               "CLKB = " + clock_name + " , " +
                               "SSRB = " + reset_name +
                          ");";
           }
}


////////////////////////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S4_S4 with aspect ratio of both ports as      //
// A: 4096 x 4 bit and B: 4096 X 4 bit                   //
////////////////////////////////////////////////////////

RESOURCEDEF RAM_MODE_S4_S4 {
```

```
////////////////////////////////
// This is connected to Ports //
// which has read/write func  //
////////////////////////////////

FUNCTIONALITY STORAGE;

////////////////////////////
// A Dual Port RAM        //
////////////////////////////

TAGS DUALPORT;

//////////////////////////////////////
// The number of addressable locations //
//////////////////////////////////////

ATTRIBUTE MAX_LOCATIONS
   {
        attribute + 4096;
   }

//////////////////////////////////////////
// The width of each addressable locations //
//////////////////////////////////////////

ATTRIBUTE MAX_WIDTH
   {
        attribute + 4;
   }

//////////////////////////////////////////
// The code to instantiate the RAM      //
// in mode S4_S4                        //
//////////////////////////////////////////

ATTRIBUTE INSTANTIATION
   {
     ////////////////////////////////////////////
     // component_name is the specific RAM that  //
     // needs to be accessed                     //
     ////////////////////////////////////////////


     ////////////////////////////////////////////
     // These signals have to be declared so that//
     // the compiler has them in the Symbol Table//
     // before it parses in the MIF interface    //
     // code from the ports.                     //
     ////////////////////////////////////////////

attribute +
```

```
                    "input unsigned fixed[12,0] " + component_name +
"_ADDRA;\n"+
                    "input unsigned fixed[4,0] " + component_name +
"_DIA;\n"+
                    "output unsigned fixed[4,0] " + component_name +
"_DOA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENA;\n";

        attribute +
                    "input unsigned fixed[12,0] " + component_name +
"_ADDRB;\n"+
                    "input unsigned fixed[4,0] " + component_name +
"_DIB;\n"+
                    "output unsigned fixed[4,0] " + component_name +
"_DOB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENB;\n";

         attribute +
            "instantiate RAMB16_S4_S4 : " + component_name +
            "(" +          "DIA = " + component_name + "_DIA," +
                         "ADDRA = " + component_name + "_ADDRA," +
                         "DOA = " + component_name + "_DOA," +
                         "ENA = " + component_name + "_ENA," +
                         "WEA = " + component_name + "_WEA," +
                               "CLKA = " + clock_name + " , " +
                               "SSRA = " + reset_name + " , " +
                           "DIB = " + component_name + "_DIB," +
                         "ADDRB = " + component_name + "_ADDRB," +
                         "DOB = " + component_name + "_DOB," +
                         "ENB = " + component_name + "_ENB," +
                         "WEB = " + component_name + "_WEB," +
                               "CLKB = " + clock_name + " , " +
                               "SSRB = " + reset_name +
                          ");";
           }
}

////////////////////////////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S9_S9 with aspect ratio of both ports as      //
// A: 2048 x 9 bit and B: 2048 X 9 bit                   //
////////////////////////////////////////////////////////////

RESOURCEDEF RAM_MODE_S9_S9 {

        ////////////////////////////////
        // This is connected to Ports //
```

```
            // which has read/write func  //
            ////////////////////////////////

            FUNCTIONALITY STORAGE;

            ////////////////////////////
            // A Dual Port RAM        //
            ////////////////////////////

            TAGS DUALPORT;

            ////////////////////////////////////////
            // The number of addressable locations //
            ////////////////////////////////////////

            ATTRIBUTE MAX_LOCATIONS
               {
                    attribute + 2048;
               }

            ////////////////////////////////////////////
            // The width of each addressable locations //
            ////////////////////////////////////////////

            ATTRIBUTE MAX_WIDTH
               {
                    attribute + 9;
               }

            ////////////////////////////////////////
            // The code to instantiate the RAM      //
            // in mode S9_S9                        //
            ////////////////////////////////////////

            ATTRIBUTE INSTANTIATION
               {
                 ////////////////////////////////////////////
                 // component_name is the specific RAM that  //
                 // needs to be accessed                     //
                 ////////////////////////////////////////////


                 ////////////////////////////////////////////
                 // These signals have to be declared so that//
                 // the compiler has them in the Symbol Table//
                 // before it parses in the MIF interface    //
                 // code from the ports.                     //
                 ////////////////////////////////////////////

            attribute +
                      "input unsigned fixed[11,0] " + component_name +
"_ADDRA;\n"+
                      "input unsigned fixed[8,0] " + component_name +
"_DIA;\n"+
```

```
                    "input unsigned fixed[1,0] " + component_name +
"_DIPA;\n"+
                    "output unsigned fixed[8,0] " + component_name +
"_DOA;\n"+
                    "output unsigned fixed[1,0] " + component_name +
"_DOPA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENA;\n";


        attribute +
                    "input unsigned fixed[11,0] " + component_name +
"_ADDRB;\n"+
                    "input unsigned fixed[8,0] " + component_name +
"_DIB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_DIPB;\n"+
                    "output unsigned fixed[8,0] " + component_name +
"_DOB;\n"+
                    "output unsigned fixed[1,0] " + component_name +
"_DOPB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENB;\n";

         attribute +
            "instantiate RAMB16_S9_S9 : " + component_name +
            "(" +          "DIA = " + component_name + "_DIA," +
                           "DIPA = " + component_name + "_DIPA," +
                         "ADDRA = " + component_name + "_ADDRA," +
                         "DOA = " + component_name + "_DOA," +
                         "DOPA = " + component_name + "_DOPA," +
                         "ENA = " + component_name + "_ENA," +
                         "WEA = " + component_name + "_WEA," +
                               "CLKA = " + clock_name + " , " +
                               "SSRA = " + reset_name + " , " +
                           "DIB = " + component_name + "_DIB," +
                           "DIPB = " + component_name + "_DIPB," +
                         "ADDRB = " + component_name + "_ADDRB," +
                         "DOB = " + component_name + "_DOB," +
                         "DOPB = " + component_name + "_DOPB," +
                         "ENB = " + component_name + "_ENB," +
                         "WEB = " + component_name + "_WEB," +
                               "CLKB = " + clock_name + " , " +
                               "SSRB = " + reset_name +
                          ");";
            }
}
```

```
//////////////////////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S18_S18 with aspect ratio of both ports as    //
// A: 1024 x 18 bit and B: 1024 X 18 bit                 //
//////////////////////////////////////////////////////

RESOURCEDEF RAM_MODE_S18_S18 {

        ////////////////////////////////
        // This is connected to Ports //
        // which has read/write func  //
        ////////////////////////////////

        FUNCTIONALITY STORAGE;

        ////////////////////////////
        // A Dual Port RAM        //
        ////////////////////////////

        TAGS DUALPORT;

        ////////////////////////////////////////
        // The number of addressable locations //
        ////////////////////////////////////////

        ATTRIBUTE MAX_LOCATIONS
           {
                attribute + 1024;
           }

        ////////////////////////////////////////////
        // The width of each addressable locations //
        ////////////////////////////////////////////

        ATTRIBUTE MAX_WIDTH
           {
                attribute + 18;
           }

        ////////////////////////////////////////
        // The code to instantiate the RAM      //
        // in mode S18_S18                      //
        ////////////////////////////////////////

        ATTRIBUTE INSTANTIATION
           {
             ////////////////////////////////////////////
             // component_name is the specific RAM that  //
             // needs to be accessed                     //
             ////////////////////////////////////////////


             ////////////////////////////////////////////
             // These signals have to be declared so that//
```

```
                // the compiler has them in the Symbol Table//
                // before it parses in the MIF interface    //
                // code from the ports.                     //
                //////////////////////////////////////////////
5
          attribute +
                    "input unsigned fixed[10,0] " + component_name +
    "_ADDRA;\n"+
                    "input unsigned fixed[16,0] " + component_name +
10  "_DIA;\n"+
                    "input unsigned fixed[2,0] " + component_name +
    "_DIPA;\n"+
                    "output unsigned fixed[16,0] " + component_name +
    "_DOA;\n"+
15                  "output unsigned fixed[2,0] " + component_name +
    "_DOPA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
    "_WEA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
20  "_ENA;\n";

          attribute +
                    "input unsigned fixed[10,0] " + component_name +
    "_ADDRB;\n"+
25                  "input unsigned fixed[16,0] " + component_name +
    "_DIB;\n"+
                    "input unsigned fixed[2,0] " + component_name +
    "_DIPB;\n"+
                    "output unsigned fixed[16,0] " + component_name +
30  "_DOB;\n"+
                    "output unsigned fixed[2,0] " + component_name +
    "_DOPB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
    "_WEB;\n"+
35                  "input unsigned fixed[1,0] " + component_name +
    "_ENB;\n";


           attribute +
40            "instantiate RAMB16_S18_S18 : " + component_name +
              "(" +          "DIA = " + component_name + "_DIA," +
                             "DIPA = " + component_name + "_DIPA," +
                           "ADDRA = " + component_name + "_ADDRA," +
                           "DOA = " + component_name + "_DOA," +
45                         "DOPA = " + component_name + "_DOPA," +
                           "ENA = " + component_name + "_ENA," +
                           "WEA = " + component_name + "_WEA," +
                                 "CLKA = " + clock_name + " , " +
                                 "SSRA = " + reset_name + " , " +
50                           "DIB = " + component_name + "_DIB," +
                             "DIPB = " + component_name + "_DIPB," +
                           "ADDRB = " + component_name + "_ADDRB," +
                           "DOB = " + component_name + "_DOB," +
                           "DOPB = " + component_name + "_DOPB," +
```

```
                        "ENB = " + component_name + "_ENB," +
                        "WEB = " + component_name + "_WEB," +
                              "CLKB = " + clock_name + " , " +
                              "SSRB = " + reset_name +
                         ");";
            }
    }


    ////////////////////////////////////////////////////////
    // A XC2V250 Block SelectRAM which is configured to work //
    // at MODE S36_S36 with aspect ratio of both ports as    //
    // A: 512 x 36 bit and B: 512 X 36 bit                   //
    ////////////////////////////////////////////////////////

    RESOURCEDEF RAM_MODE_S36_S36 {

        ////////////////////////////////
        // This is connected to Ports //
        // which has read/write func  //
        ////////////////////////////////

        FUNCTIONALITY STORAGE;

        ////////////////////////////
        // A Dual Port RAM        //
        ////////////////////////////

        TAGS DUALPORT;

        ////////////////////////////////////////
        // The number of addressable locations //
        ////////////////////////////////////////

        ATTRIBUTE MAX_LOCATIONS
           {
                attribute + 512;
           }

        ////////////////////////////////////////////
        // The width of each addressable locations //
        ////////////////////////////////////////////

        ATTRIBUTE MAX_WIDTH
           {
                attribute + 36;
           }

        ////////////////////////////////////////
        // The code to instantiate the RAM       //
        // in mode S36_S36                       //
        ////////////////////////////////////////
```

```
        ATTRIBUTE INSTANTIATION
           {
             ////////////////////////////////////////////
             // component_name is the specific RAM that  //
             // needs to be accessed                     //
             ////////////////////////////////////////////


             ////////////////////////////////////////////
             // These signals have to be declared so that//
             // the compiler has them in the Symbol Table//
             // before it parses in the MIF interface    //
             // code from the ports.                     //
             ////////////////////////////////////////////

        attribute +
                    "input unsigned fixed[9,0] " + component_name +
"_ADDRA;\n"+
                    "input unsigned fixed[32,0] " + component_name +
"_DIA;\n"+
                    "input unsigned fixed[4,0] " + component_name +
"_DIPA;\n"+
                    "output unsigned fixed[32,0] " + component_name +
"_DOA;\n"+
                    "output unsigned fixed[4,0] " + component_name +
"_DOPA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEA;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENA;\n";

         attribute +
                    "input unsigned fixed[9,0] " + component_name +
"_ADDRB;\n"+
                    "input unsigned fixed[32,0] " + component_name +
"_DIB;\n"+
                    "input unsigned fixed[4,0] " + component_name +
"_DIPB;\n"+
                    "output unsigned fixed[32,0] " + component_name +
"_DOB;\n"+
                    "output unsigned fixed[4,0] " + component_name +
"_DOPB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_WEB;\n"+
                    "input unsigned fixed[1,0] " + component_name +
"_ENB;\n";

          attribute +
             "instantiate RAMB16_S36_S36 : " + component_name +
             "(" +          "DIA = " + component_name + "_DIA," +
                            "DIPA = " + component_name + "_DIPA," +
                          "ADDRA = " + component_name + "_ADDRA," +
                          "DOA = " + component_name + "_DOA," +
```

```
                                "DOPA = " + component_name + "_DOPA," +
                                "ENA = " + component_name + "_ENA," +
                                "WEA = " + component_name + "_WEA," +
                                      "CLKA = " + clock_name + " , " +
                                      "SSRA = " + reset_name + " , " +
                                  "DIB = " + component_name + "_DIB," +
                                  "DIPB = " + component_name + "_DIPB," +
                                "ADDRB = " + component_name + "_ADDRB," +
                                "DOB = " + component_name + "_DOB," +
                                "DOPB = " + component_name + "_DOPB," +
                                "ENB = " + component_name + "_ENB," +
                                "WEB = " + component_name + "_WEB," +
                                      "CLKB = " + clock_name + " , " +
                                      "SSRB = " + reset_name +
                                 ");";
                }
    }


    ////////////////////////////////
    // A Virtual Non-Existing     //
    // Resource which connects    //
    // the memory to a memory port //
    ////////////////////////////////

    RESOURCEDEF VIRTUAL_LINK {

            ////////////////////////////////
            // This is a virtual bus      //
            ////////////////////////////////

            FUNCTIONALITY TRANSPORT;

            ////////////////////////////////
            // Tag which indicates to the //
            // compiler that the link is  //
            // virtual                    //
            ////////////////////////////////

            TAGS VIRTUAL;

            ////////////////////////////////
            // This resource can always   //
            // perform the transport      //
            // functionality              //
            ////////////////////////////////

            ATTRIBUTE CAN_DO
                {
                    attribute + "true";
                }
    }
```

```
//////////////////////////////////////////////////
// Actual Block SelectRAM resource. There are 24 of //
// these resources in the XC2V250. But since each of//
// these RAMs can be configured into 21 different   //
// modes, the RDL treats the SelectRAM operating in //
// each of the modes as a virtual resource which    //
// USES this actual resource                        //
//////////////////////////////////////////////////

RESOURCEDEF BlockSelectRAM {

        ////////////////////////////////
        // This is connected to Ports //
        // which has read/write func  //
        ////////////////////////////////

        FUNCTIONALITY STORAGE;

        ////////////////////////////
        // A Dual Port RAM        //
        ////////////////////////////

        TAGS DUALPORT;

        ////////////////////////////////
        // This resource can always   //
        // perform the transport      //
        // functionality              //
        ////////////////////////////////

        ATTRIBUTE CAN_DO
            {
                attribute + "true";
            }
}


//////////////////////////////////////////////////////////////////////////
// Functionality and Architecture description for the Xc2V250          //
// Reconfigurable part. The Logic Cells are arranged in a 24 x 16      //
// grid.                                                               //
//////////////////////////////////////////////////////////////////////////

RESOURCEDEF FPGA_ADD
{
   FUNCTIONALITY ADD;
   ATTRIBUTE CAN_DO
       {
         <ADD>
           {
              attribute + "true";
           }
```

```
        }

    ATTRIBUTE PIPE_DELAY
        {
          <ADD>
            {
               attribute + "0";
            }
        }

ATTRIBUTE COMBINATIONAL
    {
         <ADD>
           {
                 attribute + "true";
           }
    }

ATTRIBUTE NUM_STATES
    {

       <ADD>
         {
                 attribute + "1";
         }
    }

    ATTRIBUTE INTERFACE
        {
         <ADD>
                {
                    attribute + "state1 : {" + out1->name() + " = ";
                    if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                    else
                      {
                         attribute + in1->name();
                      }

                     attribute + " + " ;
                     if(in2->is_constant() == true)
                      {
                         attribute +  in2->constant_value();
                      }
                    else
                      {
                         attribute + in2->name();
                      }
                  attribute + "; \n";
                      // To end the list of states
                      attribute + "goto NEXTSTATE ;\n";
                  attribute + "}";
```

```
                }
        }
}

RESOURCEDEF FPGA_SUB
{
   FUNCTIONALITY SUB;
   ATTRIBUTE CAN_DO
       {
           <SUB>
               {
                  attribute + "true";
               }
       }

   ATTRIBUTE PIPE_DELAY
      {
        <SUB>
          {
             attribute + "0";
          }
      }

ATTRIBUTE COMBINATIONAL
   {
        <SUB>
        {
                attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {

      <SUB>
        {
                attribute + "1";
        }
   }

   ATTRIBUTE INTERFACE
       {
             <SUB>
               {
                   attribute + "state1 : {";
                   attribute + out1->name() + " = ";
                   if(in1->is_constant() == true)
                     {
                        attribute +  in1->constant_value();
                     }
                   else
                     {
                        attribute + in1->name();
                     }
```

```
                        attribute + " - " ;
                        if(in2->is_constant() == true)
                         {
                            attribute +  in2->constant_value();
                         }
                        else
                         {
                            attribute + in2->name();
                         }
                    attribute + "; \n";
                        // To end the list of states
                        attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                    }
            }
    }

    RESOURCEDEF FPGA_MULT
    {
       FUNCTIONALITY MULT;
       ATTRIBUTE CAN_DO
           {
                <MULT>
                   {
                      attribute + "true";
                   }
           }

       ATTRIBUTE PIPE_DELAY
           {
             <MULT>
               {
                  attribute + "0";
               }
           }

    ATTRIBUTE COMBINATIONAL
       {
            <MULT>
            {
                    attribute + "true";
            }
       }

    ATTRIBUTE NUM_STATES
       {

          <MULT>
            {
                    attribute + "1";
            }
       }
```

```
    ATTRIBUTE INTERFACE
        {
            <MULT>
                {
                    attribute + "state1 : {";
                    attribute + out1->name() + " = ";
                    if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                    else
                      {
                         attribute + in1->name();
                      }

                     attribute + " * " ;
                     if(in2->is_constant() == true)
                      {
                         attribute +  in2->constant_value();
                      }
                    else
                      {
                         attribute + in2->name();
                      }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                }
        }
}

RESOURCEDEF FPGA_DIV
{
   FUNCTIONALITY DIV;
   ATTRIBUTE CAN_DO
        {
             <DIV>
                {
                   if(in2->is_constant() == true &&
                    in2->is_power_of_two() == true)
                      {
                         attribute + "true";
                      }
                    else
                      {
                         attribute + "false";
                      }
                 }
        }

   ATTRIBUTE PIPE_DELAY
      {
        <DIV>
```

```
            {
               attribute + "0";
            }
        }

ATTRIBUTE COMBINATIONAL
   {
        <DIV>
        {
                attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {

      <DIV>
        {
                attribute + "1";
        }
   }

   ATTRIBUTE INTERFACE
       {
             <DIV>
                 {
                   attribute + "state1 : {";
                   attribute + out1->name() + " = ";
                   if(in1->is_constant() == true)
                     {
                        attribute +  in1->constant_value();
                     }
                   else
                     {
                        attribute + in1->name();
                     }

                    attribute + " / " ;
                    if(in2->is_constant() == true)
                     {
                        attribute +  in2->constant_value();
                     }
                   else
                     {
                        attribute + in2->name();
                     }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}
```

```
RESOURCEDEF FPGA_REM
{
   FUNCTIONALITY REM;
   ATTRIBUTE CAN_DO
       {
              <REM>
               {
                  if(in2->is_constant() == true &&
                   in2->is_power_of_two() == true &&
                   in1->bitwidth() == out1->bitwidth())
                     {
                       attribute + "true";
                     }
                   else
                     {
                        attribute + "false";
                     }
               }
       }

   ATTRIBUTE PIPE_DELAY
      {
        <REM>
          {
             attribute + "0";
          }
      }

ATTRIBUTE COMBINATIONAL
   {
        <REM>
        {
                attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {

      <REM>
        {
                attribute + "1";
        }
   }

   ATTRIBUTE INTERFACE
       {
              <REM>
                 {
                   attribute + "state1 : {";
                   attribute + out1->name() + " = ";
                   if(in1->is_constant() == true)
                     {
                        attribute +  in1->constant_value();
```

```
                    }
                  else
                    {
                       attribute + in1->name();
                    }

                    attribute + " rem " ;
                    if(in2->is_constant() == true)
                    {
                       attribute +  in2->constant_value();
                    }
                  else
                    {
                       attribute + in2->name();
                    }
               attribute + "; \n";
                   // To end the list of states
                   attribute + "goto NEXTSTATE ;\n";
               attribute + "}";
                  }
        }
    }

RESOURCEDEF FPGA_AND
{
   FUNCTIONALITY AND;
   ATTRIBUTE CAN_DO
       {
             <AND>
               {
                if(in1->bitwidth() == in2->bitwidth() &&
                 out1->bitwidth() == in1->bitwidth())
                {
                  attribute + "true";
                }
               else
                   {
                   attribute + "false";
                }
               }
        }

    ATTRIBUTE PIPE_DELAY
       {
         <AND>
           {
              attribute + "0";
           }
       }

ATTRIBUTE COMBINATIONAL
   {
        <AND>
```

```
            {
                    attribute + "true";
            }
        }

ATTRIBUTE NUM_STATES
    {

        <AND>
          {
                    attribute + "1";
          }
    }

    ATTRIBUTE INTERFACE
        {
                <AND>
                   {
                     attribute + "state1 : {";
                     attribute + out1->name() + " = ";
                     if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                     else
                      {
                         attribute + in1->name();
                      }

                     attribute + " and " ;
                     if(in2->is_constant() == true)
                      {
                         attribute +  in2->constant_value();
                      }
                     else
                      {
                         attribute + in2->name();
                      }
                 attribute + "; \n";
                     // To end the list of states
                     attribute + "goto NEXTSTATE ;\n";
                 attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_OR
{
   FUNCTIONALITY OR;
   ATTRIBUTE CAN_DO
        {
              <OR>
                {
                 if(in1->bitwidth() == in2->bitwidth() &&
```

```
                    out1->bitwidth() == in1->bitwidth())
                   {
                     attribute + "true";
                   }
                 else
                     {
                     attribute + "false";
                   }
                  }
         }

    ATTRIBUTE PIPE_DELAY
       {
         <OR>
           {
              attribute + "0";
           }
       }

ATTRIBUTE COMBINATIONAL
    {
         <OR>
         {
                 attribute + "true";
         }
    }

ATTRIBUTE NUM_STATES
    {

       <OR>
         {
                 attribute + "1";
         }
    }

    ATTRIBUTE INTERFACE
        {
                <OR>
                   {
                     attribute + "state1 : {";
                     attribute + out1->name() + " = ";
                     if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                     else
                      {
                         attribute + in1->name();
                      }

                     attribute + " or " ;
                     if(in2->is_constant() == true)
                      {
```

```
                        attribute +  in2->constant_value();
                    }
                   else
                    {
                        attribute + in2->name();
                    }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_NOT
{
   FUNCTIONALITY NOT;
   ATTRIBUTE CAN_DO
       {
             <NOT>
               {
                  if(in1->is_integer() == true)
                    {
                        attribute + "false";
                    }
                   else
                    {
                        attribute + "true";
                    }
               }
        }

    ATTRIBUTE PIPE_DELAY
       {
         <NOT>
           {
              attribute + "0";
           }
       }

 ATTRIBUTE COMBINATIONAL
    {
         <NOT>
         {
                 attribute + "true";
         }
    }

 ATTRIBUTE NUM_STATES
    {

        <NOT>
          {
                  attribute + "1";
```

```
        }
    }

    ATTRIBUTE INTERFACE
        {
                <NOT>
                   {
                     attribute + "state1 : {";
                     attribute + out1->name() + " = ";
                     attribute + " ! " ;
                     if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                     else
                      {
                         attribute + in1->name();
                      }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                    }
        }
}

RESOURCEDEF FPGA_XOR
{
   FUNCTIONALITY XOR;
   ATTRIBUTE CAN_DO
        {
              <XOR>
                {
                 if(in1->bitwidth() == in2->bitwidth() &&
                  out1->bitwidth() == in1->bitwidth())
                 {
                   attribute + "true";
                 }
                else
                    {
                    attribute + "false";
                 }
                }
        }

   ATTRIBUTE PIPE_DELAY
      {
        <XOR>
          {
             attribute + "0";
          }
      }

ATTRIBUTE COMBINATIONAL
```

```
        {
              <XOR>
              {
                      attribute + "true";
              }
        }

    ATTRIBUTE NUM_STATES
        {

           <XOR>
             {
                      attribute + "1";
             }
        }

        ATTRIBUTE INTERFACE
            {
                    <XOR>
                       {
                         attribute + "state1 : {";
                         attribute + out1->name() + " = ";
                         if(in1->is_constant() == true)
                          {
                             attribute +  in1->constant_value();
                          }
                         else
                          {
                             attribute + in1->name();
                          }

                         attribute + " xor " ;
                         if(in2->is_constant() == true)
                          {
                             attribute +  in2->constant_value();
                          }
                         else
                          {
                             attribute + in2->name();
                          }
                       attribute + "; \n";
                           // To end the list of states
                           attribute + "goto NEXTSTATE ;\n";
                       attribute + "}";
                          }
            }
    }

    RESOURCEDEF FPGA_EQ
    {
       FUNCTIONALITY EQ;
       ATTRIBUTE CAN_DO
           {
                  <EQ>
```

```
                {
                if(in1->is_integer() == true && in2->is_integer() == true
&&
                     out1->is_boolean() == true)
                     {
                         attribute + "true";
                     }
                  else
                     {
                       if(in1->is_fixed() == true && in2->is_fixed() ==
true &&
                          out1->is_boolean() == true)
                         {
                           attribute + "true";
                         }
                        else
                         {
                           if(in1->is_constant() == true ||
                              in2->is_constant() == true &&
                              out1->is_boolean() == true)
                             {
                                 attribute + "true";
                             }
                            else
                             {
                               attribute + "false";
                             }
                         }
                     }
                }
          }

    ATTRIBUTE PIPE_DELAY
       {
         <EQ>
           {
              attribute + "0";
           }
       }

ATTRIBUTE COMBINATIONAL
   {
        <EQ>
        {
                attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {

      <EQ>
        {
                attribute + "1";
```

```
        }
    }

    ATTRIBUTE INTERFACE
        {
                <EQ>
                  {
                    attribute + "state1 : {";
                    attribute + out1->name() + " = ";
                    if(in1->is_constant() == true)
                     {
                        attribute +  in1->constant_value();
                     }
                    else
                     {
                        attribute + in1->name();
                     }

                    attribute + " == " ;
                    if(in2->is_constant() == true)
                     {
                        attribute +  in2->constant_value();
                     }
                    else
                     {
                        attribute + in2->name();
                     }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_NEQ
{
   FUNCTIONALITY NEQ;
   ATTRIBUTE CAN_DO
        {
             <NEQ>
                {
                  if(in1->is_integer() == true && in2->is_integer() ==
true &&
                     out1->is_boolean() == true)
                     {
                         attribute + "true";
                     }
                  else
                     {
                       if(in1->is_fixed() == true && in2->is_fixed() ==
true &&
                          out1->is_boolean() == true)
                         {
```

```
                        attribute + "true";
                      }
                    else
                      {
                        attribute + "false";
                      }
                  }
              }

    ATTRIBUTE PIPE_DELAY
       {
         <NEQ>
           {
              attribute + "0";
           }
       }

ATTRIBUTE COMBINATIONAL
   {
        <NEQ>
        {
                attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {
      <NEQ>
        {
                attribute + "1";
        }
   }

   ATTRIBUTE INTERFACE
       {
              <NEQ>
                 {
                   attribute + "state1 : {";
                   attribute + out1->name() + " = ";
                   if(in1->is_constant() == true)
                    {
                       attribute +  in1->constant_value();
                    }
                   else
                    {
                       attribute + in1->name();
                    }

                   attribute + " \\= " ;
                   if(in2->is_constant() == true)
                    {
                       attribute +  in2->constant_value();
                    }
```

```
                    else
                     {
                        attribute + in2->name();
                     }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_GT
{
   FUNCTIONALITY GT;
   ATTRIBUTE CAN_DO
       {
              <GT>
              {
                 if(out1->is_boolean() == true)
                   {
                        attribute + "true";
                   }
                 else
                   {
                        attribute + "false";
                   }
              }
       }

    ATTRIBUTE PIPE_DELAY
       {
         <GT>
           {
              attribute + "0";
           }
       }

ATTRIBUTE COMBINATIONAL
   {
        <GT>
        {
               attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {
      <GT>
        {
               attribute + "1";
        }
   }
```

```
    ATTRIBUTE INTERFACE
        {
                <GT>
                   {
                     attribute + "state1 : {";
                     attribute + out1->name() + " = ";
                     if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                     else
                      {
                         attribute + in1->name();
                      }

                     attribute + " > " ;
                     if(in2->is_constant() == true)
                      {
                         attribute +  in2->constant_value();
                      }
                     else
                      {
                         attribute + in2->name();
                      }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_GE
{
   FUNCTIONALITY GE;
   ATTRIBUTE CAN_DO
        {
               <GE>
               {
                  if(out1->is_boolean() == true)
                    {
                         attribute + "true";
                    }
                  else
                    {
                         attribute + "false";
                    }
               }
        }

   ATTRIBUTE PIPE_DELAY
      {
        <GE>
          {
```

```
                attribute + "0";
            }
        }

ATTRIBUTE COMBINATIONAL
    {
        <GE>
        {
                attribute + "true";
        }
    }

ATTRIBUTE NUM_STATES
    {
       <GE>
        {
                attribute + "1";
        }
    }

    ATTRIBUTE INTERFACE
        {
                <GE>
                   {
                     attribute + "state1 : {";
                     attribute + out1->name() + " = ";
                     if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                     else
                      {
                         attribute + in1->name();
                      }

                     attribute + " >= " ;
                     if(in2->is_constant() == true)
                      {
                         attribute +  in2->constant_value();
                      }
                     else
                      {
                         attribute + in2->name();
                      }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_LT
{
```

```
FUNCTIONALITY LT;
ATTRIBUTE CAN_DO
    {
        <LT>
          {
              if(out1->is_boolean() == true)
               {
                    attribute + "true";
               }
              else
               {
                    attribute + "false";
               }
          }
    }

ATTRIBUTE PIPE_DELAY
   {
     <LT>
       {
          attribute + "0";
       }
   }

ATTRIBUTE COMBINATIONAL
   {
        <LT>
        {
                attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {
      <LT>
        {
                attribute + "1";
        }
   }

ATTRIBUTE INTERFACE
     {
            <LT>
               {
                 attribute + "state1 : {";
                 attribute + out1->name() + " = ";
                 if(in1->is_constant() == true)
                  {
                     attribute +  in1->constant_value();
                  }
                 else
                  {
                     attribute + in1->name();
                  }
```

```
                    attribute + " < " ;
                    if(in2->is_constant() == true)
                     {
                        attribute +  in2->constant_value();
                     }
                    else
                     {
                        attribute + in2->name();
                     }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_LE
{
   FUNCTIONALITY LE;
   ATTRIBUTE CAN_DO
       {
            <LE>
              {
                  if(out1->is_boolean() == true)
                   {
                        attribute + "true";
                   }
                 else
                   {
                        attribute + "false";
                   }
              }
       }

    ATTRIBUTE PIPE_DELAY
       {
         <LE>
           {
              attribute + "0";
           }
       }

ATTRIBUTE COMBINATIONAL
   {
        <LE>
        {
                attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {
```

```
        <LE>
          {
                  attribute + "1";
          }
    }

    ATTRIBUTE INTERFACE
        {
                <LE>
                   {
                     attribute + "state1 : {";
                     attribute + out1->name() + " = ";
                     if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                     else
                      {
                         attribute + in1->name();
                      }

                     attribute + " <= " ;
                     if(in2->is_constant() == true)
                      {
                         attribute +  in2->constant_value();
                      }
                     else
                      {
                         attribute + in2->name();
                      }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_SLL
{
   FUNCTIONALITY SLL;
   ATTRIBUTE CAN_DO
        {
             <SLL>
              {
                 if(in2->is_integer() == true && in1->is_fixed() == true
&&
                    out1->is_fixed() == true &&
                  in2->is_power_of_two() == true &&
                  in1->bitwidth() == out1->bitwidth())
                   {
                         attribute + "true";
                   }
                 else
```

```
                    {
                        attribute + "false";
                    }
                }
        }

    ATTRIBUTE PIPE_DELAY
        {
          <SLL>
            {
                attribute + "0";
            }
        }

ATTRIBUTE COMBINATIONAL
    {
         <SLL>
         {
                attribute + "true";
         }
    }

    ATTRIBUTE NUM_STATES
        {
           <SLL>
             {
                attribute + "1";
             }
        }

        ATTRIBUTE INTERFACE
            {
                <SLL>
                  {
                    attribute + "state1 : {";
                    attribute + out1->name() + " = ";
                    if(in1->is_constant() == true)
                     {
                        attribute +  in1->constant_value();
                     }
                    else
                     {
                        attribute + in1->name();
                     }

                    attribute + " sll " ;
                    if(in2->is_constant() == true)
                     {
                        attribute +  in2->constant_value();
                     }
                    else
                     {
                        attribute + in2->name();
                     }
```

```
                    attribute + "; \n";
                        // To end the list of states
                        attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                       }
            }
    }

    RESOURCEDEF FPGA_SLR
    {
       FUNCTIONALITY SLR;
       ATTRIBUTE CAN_DO
           {
               <SLR>
                 {
                    if(in2->is_integer() == true && in1->is_fixed() == true
    &&
                       out1->is_fixed() == true &&
                     in2->is_power_of_two() == true &&
                     in1->bitwidth() == out1->bitwidth())
                      {
                            attribute + "true";
                      }
                    else
                      {
                            attribute + "false";
                      }
                 }
           }

       ATTRIBUTE PIPE_DELAY
          {
            <SLR>
              {
                 attribute + "0";
              }
          }

    ATTRIBUTE COMBINATIONAL
       {
            <SLR>
            {
                    attribute + "true";
            }
       }

    ATTRIBUTE NUM_STATES
       {
          <SLR>
            {
                    attribute + "1";
            }
       }
```

```
ATTRIBUTE INTERFACE
    {
            <SLR>
               {
                 attribute + "state1 : {";
                 attribute + out1->name() + " = ";
                 if(in1->is_constant() == true)
                  {
                     attribute +  in1->constant_value();
                  }
                 else
                  {
                     attribute + in1->name();
                  }

                 attribute + " srl " ;
                 if(in2->is_constant() == true)
                  {
                     attribute +  in2->constant_value();
                  }
                 else
                  {
                     attribute + in2->name();
                  }
              attribute + "; \n";
                 // To end the list of states
                 attribute + "goto NEXTSTATE ;\n";
              attribute + "}";
               }
        }
}

RESOURCEDEF FPGA_SLA
{
   FUNCTIONALITY SLA;
   ATTRIBUTE CAN_DO
       {
           <SLA>
              {
                if(in2->is_integer() == true && in1->is_fixed() == true
&&
                   out1->is_fixed() == true &&
                 in2->is_power_of_two() == true &&
                 in1->bitwidth() == out1->bitwidth())
                  {
                       attribute + "true";
                  }
                else
                  {
                       attribute + "false";
                  }
              }
       }
```

```
   ATTRIBUTE PIPE_DELAY
      {
        <SLA>
          {
             attribute + "0";
          }
      }

ATTRIBUTE COMBINATIONAL
   {
        <SLA>
        {
                attribute + "true";
        }
   }

   ATTRIBUTE NUM_STATES
   {
      <SLA>
        {
                attribute + "1";
        }
   }

   ATTRIBUTE INTERFACE
       {
               <SLA>
                  {
                    attribute + "state1 : {";
                    attribute + out1->name() + " = ";
                    if(in1->is_constant() == true)
                     {
                        attribute +  in1->constant_value();
                     }
                    else
                     {
                        attribute + in1->name();
                     }

                    attribute + " sla " ;
                    if(in2->is_constant() == true)
                     {
                        attribute +  in2->constant_value();
                     }
                    else
                     {
                        attribute + in2->name();
                     }
               attribute + "; \n";
                   // To end the list of states
                   attribute + "goto NEXTSTATE ;\n";
               attribute + "}";
                  }
```

```
        }
    }

    RESOURCEDEF FPGA_SRA
5   {
       FUNCTIONALITY SRA;
       ATTRIBUTE CAN_DO
           {
               <SRA>
                  {
10                  if(in2->is_integer() == true && in1->is_fixed() == true
    &&
                        out1->is_fixed() == true &&
                      in2->is_power_of_two() == true &&
                      in1->bitwidth() == out1->bitwidth())
15                     {
                             attribute + "true";
                       }
                     else
                       {
20                           attribute + "false";
                       }
                  }
           }
25
       ATTRIBUTE PIPE_DELAY
           {
             <SRA>
               {
30               attribute + "0";
               }
           }

     ATTRIBUTE COMBINATIONAL
35      {
             <SRA>
             {
                     attribute + "true";
             }
40      }

     ATTRIBUTE NUM_STATES
        {
           <SRA>
45           {
                     attribute + "1";
             }
        }

50      ATTRIBUTE INTERFACE
            {
                    <SRA>
                       {
                         attribute + "state1 : {";
```

```
                        attribute + out1->name() + " = ";
                        if(in1->is_constant() == true)
                         {
                            attribute +  in1->constant_value();
                         }
                        else
                         {
                            attribute + in1->name();
                         }

                        attribute + " sra " ;
                        if(in2->is_constant() == true)
                         {
                            attribute +  in2->constant_value();
                         }
                        else
                         {
                            attribute + in2->name();
                         }
                    attribute + "; \n";
                        // To end the list of states
                        attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                       }
            }
    }

    RESOURCEDEF FPGA_ROL
    {
       FUNCTIONALITY ROL;
       ATTRIBUTE CAN_DO
            {
                <ROL>
                 {
                     if(in2->is_integer() == true && in1->is_fixed() == true
    &&
                        out1->is_fixed() == true &&
                      in2->is_power_of_two() == true &&
                      in1->bitwidth() == out1->bitwidth())
                       {
                            attribute + "true";
                       }
                     else
                       {
                            attribute + "false";
                       }
                 }
            }

       ATTRIBUTE PIPE_DELAY
          {
            <ROL>
              {
                 attribute + "0";
```

```
            }
        }

    ATTRIBUTE COMBINATIONAL
    {
        <ROL>
        {
                attribute + "true";
        }
    }

    ATTRIBUTE NUM_STATES
    {
       <ROL>
        {
                attribute + "1";
        }
    }

    ATTRIBUTE INTERFACE
        {
                <ROL>
                   {
                     attribute + "state1 : {";
                     attribute + out1->name() + " = ";
                     if(in1->is_constant() == true)
                      {
                         attribute +  in1->constant_value();
                      }
                     else
                      {
                         attribute + in1->name();
                      }

                     attribute + " rol " ;
                     if(in2->is_constant() == true)
                      {
                         attribute +  in2->constant_value();
                      }
                     else
                      {
                         attribute + in2->name();
                      }
                attribute + "; \n";
                    // To end the list of states
                    attribute + "goto NEXTSTATE ;\n";
                attribute + "}";
                   }
        }
}

RESOURCEDEF FPGA_ROR
{
   FUNCTIONALITY ROR;
```

```
    ATTRIBUTE CAN_DO
        {
            <ROR>
             {
                 if(in2->is_integer() == true && in1->is_fixed() == true
&&
                    out1->is_fixed() == true &&
                   in2->is_power_of_two() == true &&
                   in1->bitwidth() == out1->bitwidth())
                    {
                          attribute + "true";
                    }
                 else
                    {
                          attribute + "false";
                    }
             }
        }

    ATTRIBUTE PIPE_DELAY
       {
         <ROR>
           {
              attribute + "0";
           }
       }

ATTRIBUTE COMBINATIONAL
   {
        <ROR>
        {
                attribute + "true";
        }
   }

ATTRIBUTE NUM_STATES
   {
      <ROR>
        {
                attribute + "1";
        }
   }

   ATTRIBUTE INTERFACE
       {
              <ROR>
                 {
                   attribute + "state1 : {";
                   attribute + out1->name() + " = ";
                   if(in1->is_constant() == true)
                    {
                       attribute +  in1->constant_value();
                    }
                   else
```

```
                        {
                           attribute + in1->name();
                        }

                        attribute + " ror " ;
                        if(in2->is_constant() == true)
                        {
                           attribute +  in2->constant_value();
                        }
                        else
                        {
                           attribute + in2->name();
                        }
                    attribute + "; \n";
                        // To end the list of states
                        attribute + "goto NEXTSTATE ;\n";
                    attribute + "}";
                       }
            }
}

RESOURCEDEF FPGA
{
   FUNCTIONALITY RECONFIG;

   ////////////////////////
   // Total number of CLBs //
   ////////////////////////
   ATTRIBUTE AREA
     {
        attribute + "4032";
     }
}

////////////////////////////
// XC2V250 Architecture    //
////////////////////////////

UNITDEF XC2V250 {

        ////////////////////////////////////
        // This is not an actual resource //
        // but rather a placeholder for   //
        // component declarations         //
        ////////////////////////////////////

        RESOURCE GLOBAL declarations;
        RESOURCE UNSIGNED_DIVIDER udiv;
        RESOURCE SIGNED_DIVIDER sdiv;

        ////////////////////////////////////
        // The embedded multipliers       //
        ////////////////////////////////////
```

```
        RESOURCE MULT18X18 embedmults[ NUM_MULT_COLUMNS *
                                NUM_EMBED_MULT_PER_COLUMN ];

        ////////////////////////////////////
        // The synchronous embedded       //
      // multipliers                    //
        ////////////////////////////////////

        RESOURCE MULT18X18S syn_embedmults[ NUM_MULT_COLUMNS *
                                NUM_EMBED_MULT_PER_COLUMN ];

        ////////////////////////////////
        // The Reconfigurable array   //
        // in the X2V250              //
        ////////////////////////////////

        RESOURCE FPGA fpga_part;

        ////////////////////////////////
        // List of resources          //
        // which use the FPGA         //
        ////////////////////////////////

        RESOURCE FPGA_ADD fpga_add;
        RESOURCE FPGA_SUB fpga_sub;
        RESOURCE FPGA_MULT fpga_mult;
        RESOURCE FPGA_DIV fpga_div;
        RESOURCE FPGA_REM fpga_rem;
        RESOURCE FPGA_AND fpga_and;
        RESOURCE FPGA_OR fpga_or;
        RESOURCE FPGA_NOT fpga_not;
        RESOURCE FPGA_XOR fpga_xor;
        RESOURCE FPGA_EQ fpga_eq;
        RESOURCE FPGA_NEQ fpga_neq;
        RESOURCE FPGA_GT fpga_gt;
        RESOURCE FPGA_LT fpga_lt;
        RESOURCE FPGA_GE fpga_ge;
        RESOURCE FPGA_LE fpga_le;
        RESOURCE FPGA_SLL fpga_sll;
        RESOURCE FPGA_SLR fpga_slr;
        RESOURCE FPGA_SLA fpga_sla;
        RESOURCE FPGA_SRA fpga_sra;
        RESOURCE FPGA_ROL fpga_rol;
        RESOURCE FPGA_ROR fpga_ror;

        ////////////////////////////////////
        // The 24 Block SelectRAMs in the //
        // XC2V250                        //
        ////////////////////////////////////

        RESOURCE BlockSelectRAM block_select_ram[NUM_RAM];

        ////////////////////////////////////////////////////////////
        // A XC2V250 Block SelectRAM which is configured to work //
```

```
        // at MODE S18_S18 with aspect ratio of both ports as    //
        // A: 16.384 x 1 bit and B: 16,384 X 1 bit               //
        ////////////////////////////////////////////////////////////

5       RESOURCE RAM_MODE_S1_S1 ram_s1_s1[NUM_RAM];
        RESOURCE RAM_PORT_A_S1_S1 porta_s1_s1[NUM_RAM];
        RESOURCE RAM_PORT_B_S1_S1 portb_s1_s1[NUM_RAM];
        RESOURCE VIRTUAL_LINK vlinka_s1_s1[NUM_RAM];
        RESOURCE VIRTUAL_LINK vlinkb_s1_s1[NUM_RAM];
10

        ////////////////////////////////////////////////////////////
        // A XC2V250 Block SelectRAM which is configured to work //
        // at MODE S18_S18 with aspect ratio of both ports as    //
15      // A: 8192 x 2 bit and B: 8192 X 2 bit                   //
        ////////////////////////////////////////////////////////////

        RESOURCE RAM_MODE_S2_S2 ram_s2_s2[NUM_RAM];
        RESOURCE RAM_PORT_A_S2_S2 porta_s2_s2[NUM_RAM];
20      RESOURCE RAM_PORT_B_S2_S2 portb_s2_s2[NUM_RAM];
        RESOURCE VIRTUAL_LINK vlinka_s2_s2[NUM_RAM];
        RESOURCE VIRTUAL_LINK vlinkb_s2_s2[NUM_RAM];


25      ////////////////////////////////////////////////////////////
        // A XC2V250 Block SelectRAM which is configured to work //
        // at MODE S18_S18 with aspect ratio of both ports as    //
        // A: 4096 x 4 bit and B: 4096 X 4 bit                   //
        ////////////////////////////////////////////////////////////
30

        RESOURCE RAM_MODE_S4_S4 ram_s4_s4[NUM_RAM];
        RESOURCE RAM_PORT_A_S4_S4 porta_s4_s4[NUM_RAM];
        RESOURCE RAM_PORT_B_S4_S4 portb_s4_s4[NUM_RAM];
        RESOURCE VIRTUAL_LINK vlinka_s4_s4[NUM_RAM];
35      RESOURCE VIRTUAL_LINK vlinkb_s4_s4[NUM_RAM];

        ////////////////////////////////////////////////////////////
        // A XC2V250 Block SelectRAM which is configured to work //
        // at MODE S18_S18 with aspect ratio of both ports as    //
40      // A: 2048 x 9 bit and B: 2048 X 9 bit                   //
        ////////////////////////////////////////////////////////////

        RESOURCE RAM_MODE_S9_S9 ram_s9_s9[NUM_RAM];
        RESOURCE RAM_PORT_A_S9_S9 porta_s9_s9[NUM_RAM];
45      RESOURCE RAM_PORT_B_S9_S9 portb_s9_s9[NUM_RAM];
        RESOURCE VIRTUAL_LINK vlinka_s9_s9[NUM_RAM];
        RESOURCE VIRTUAL_LINK vlinkb_s9_s9[NUM_RAM];


50      ////////////////////////////////////////////////////////////
        // A XC2V250 Block SelectRAM which is configured to work //
        // at MODE S18_S18 with aspect ratio of both ports as    //
        // A: 1024 x 18 bit and B: 1024 X 18 bit                 //
        ////////////////////////////////////////////////////////////
```

```
RESOURCE RAM_MODE_S18_S18 ram_s18_s18[NUM_RAM];
RESOURCE RAM_PORT_A_S18_S18 porta_s18_s18[NUM_RAM];
RESOURCE RAM_PORT_B_S18_S18 portb_s18_s18[NUM_RAM];
RESOURCE VIRTUAL_LINK vlinka_s18_s18[NUM_RAM];
RESOURCE VIRTUAL_LINK vlinkb_s18_s18[NUM_RAM];


////////////////////////////////////////////////////////
// A XC2V250 Block SelectRAM which is configured to work //
// at MODE S36_S36 with aspect ratio of both ports as    //
// A: 512 x 36 bit and B: 512 X 36 bit                   //
////////////////////////////////////////////////////////

RESOURCE RAM_MODE_S36_S36 ram_s36_s36[NUM_RAM];
RESOURCE RAM_PORT_A_S36_S36 porta_s36_s36[NUM_RAM];
RESOURCE RAM_PORT_B_S36_S36 portb_s36_s36[NUM_RAM];
RESOURCE VIRTUAL_LINK vlinka_s36_s36[NUM_RAM];
RESOURCE VIRTUAL_LINK vlinkb_s36_s36[NUM_RAM];

////////////////////////////////////////////////////////
// The RAM resource at various modes is always connected //
// to 2 ports. Since this connection is always present   //
// as the RAM implies its 2 ports, we have created a     //
// virtual routing resource for connecting the RAm to    //
// its ports.                                            //
////////////////////////////////////////////////////////

for(i = 0;i < NUM_RAM;i = i + 1)
{
   RCONNECT(ram_s1_s1[i],porta_s1_s1[i],vlinka_s1_s1[i]);
   RCONNECT(ram_s1_s1[i],portb_s1_s1[i],vlinkb_s1_s1[i]);
}

for(i = 0;i < NUM_RAM;i = i + 1)
{
   RCONNECT(ram_s2_s2[i],porta_s2_s2[i],vlinka_s2_s2[i]);
   RCONNECT(ram_s2_s2[i],portb_s2_s2[i],vlinkb_s2_s2[i]);
}

for(i = 0;i < NUM_RAM;i = i + 1)
{
   RCONNECT(ram_s4_s4[i],porta_s4_s4[i],vlinka_s4_s4[i]);
   RCONNECT(ram_s4_s4[i],portb_s4_s4[i],vlinkb_s4_s4[i]);
}

for(i = 0;i < NUM_RAM;i = i + 1)
{
   RCONNECT(ram_s9_s9[i],porta_s9_s9[i],vlinka_s9_s9[i]);
   RCONNECT(ram_s9_s9[i],portb_s9_s9[i],vlinkb_s9_s9[i]);
}

for(i = 0;i < NUM_RAM;i = i + 1)
{
```

```
            RCONNECT(ram_s18_s18[i],porta_s18_s18[i],vlinka_s18_s18[i]);
            RCONNECT(ram_s18_s18[i],portb_s18_s18[i],vlinkb_s18_s18[i]);
        }

        for(i = 0;i < NUM_RAM;i = i + 1)
        {
            RCONNECT(ram_s36_s36[i],porta_s36_s36[i],vlinka_s36_s36[i]);
            RCONNECT(ram_s36_s36[i],portb_s36_s36[i],vlinkb_s36_s36[i]);
        }


        ////////////////////////////////////////
        // The different RAM configurations   //
        // are actually virtual resources which //
        // use the actual SelectRAM resource  //
        ////////////////////////////////////////


        for(i = 0;i < NUM_RAM;i = i + 1)
            {
                USES(ram_s1_s1[i],block_select_ram[i]);
            }

        for(i = 0;i < NUM_RAM;i = i + 1)
            {
                USES(ram_s2_s2[i],block_select_ram[i]);
            }

        for(i = 0;i < NUM_RAM;i = i + 1)
            {
                USES(ram_s4_s4[i],block_select_ram[i]);
            }

        for(i = 0;i < NUM_RAM;i = i + 1)
            {
                USES(ram_s9_s9[i],block_select_ram[i]);
            }

        for(i = 0;i < NUM_RAM;i = i + 1)
            {
                USES(ram_s18_s18[i],block_select_ram[i]);
            }

        for(i = 0;i < NUM_RAM;i = i + 1)
            {
                USES(ram_s36_s36[i],block_select_ram[i]);
            }

      ////////////////////////////////////////////
      // The synchronous embedded multipliers are //
      // modelled as virtual resources using    //
      // the actual physical multipliers      //
      ////////////////////////////////////////////
```

```
        for(i = 0;i< NUM_MULT_COLUMNS * NUM_EMBED_MULT_PER_COLUMN;i = i +
    1)
        {
            USES(syn_embedmults[i],embedmults[i]);
5       }
    }
```